

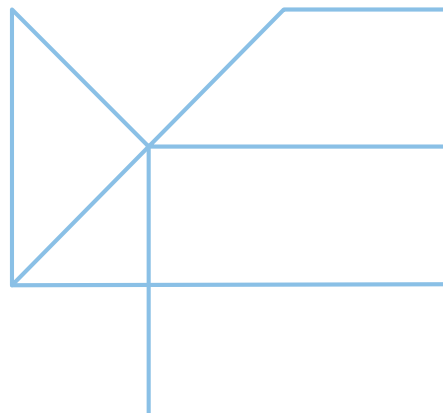
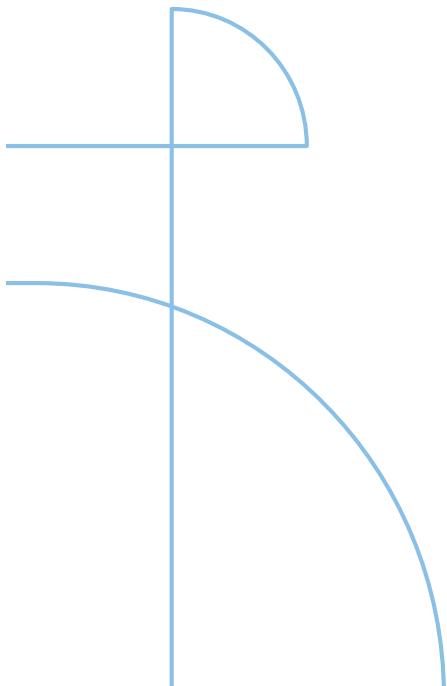


Licentiate Thesis in Decision and Control Systems

Towards Efficient and Robust Decentralized Learning

ZESEN WANG

KTH ROYAL INSTITUTE OF TECHNOLOGY



Towards Efficient and Robust Decentralized Learning

ZESEN WANG

Academic Dissertation which, with due permission of the KTH Royal Institute of Technology, is submitted for public defence for the Degree of Licentiate of Technology in Electrical Engineering on Thursday, the 5th March 2026, at 3:00 PM in D3, Lindstedtsvägen 5, Kungliga Tekniska Högskolan, Stockholm.

Licentiate Thesis in Decision and Control Systems
KTH Royal Institute of Technology
Stockholm, Sweden 2026

© Zesen Wang 2026

All Rights Reserved. No part of this work may be reproduced in any form without the written permission of the copyright owner.

TRITA-EECS-AVL-2026:13
ISBN 978-91-8106-520-6

Printed by Universitetservice US-AB, Sweden 2026

Abstract

The widening gap between GPU compute capability and inter-node network bandwidth presents a fundamental challenge for distributed deep learning. While traditional "All-Reduce" methods require every GPU to sync globally — slowing down the entire system to the speed of the slowest worker — decentralized training allows GPUs to communicate only with a few neighbors. Despite its potential, decentralized training is rarely adopted in practice because its performance gains are hard to predict, its impact on model accuracy is poorly understood, and it is complex to implement.

This thesis investigates Decentralized Training as a robust and efficient alternative to global synchronization. By restricting communication to a sparse graph of neighbors, decentralized algorithms reduce bandwidth usage and alleviate the single point of straggler inherent in global collective communications. Despite these theoretical advantages, adoption has been hindered by three key challenges: ambiguity regarding efficiency gains, uncertainty about generalization performance, and implementation barriers.

To address the efficiency ambiguity, we propose a comprehensive *runtime model* that characterizes the capability of decentralized algorithms. We derive an analytical bound that characterizes hardware–model regimes under which decentralized training can outperform the All-Reduce method by a margin, validating this model on GPU clusters. This analysis highlights the relevance of decentralized schemes as the "outer loop" synchronization mechanism in bandwidth-constrained environments.

Second, we tackle the generalization uncertainty by analyzing the role of consensus error. We initially propose *AccumAdam*, an engineering stabilization mechanism designed to mitigate momentum drift caused by decentralization and stabilize convergence. We then pivot to a novel perspective with *DSGD-AC* (Adaptive Consensus), demonstrating that consensus error — often viewed as harmful noise —

can act as an implicit regularization mechanism related to curvature. We show that by controlling rather than eliminating this error, decentralized training can favor smooth minima and improve generalization compared to centralized baselines.

Finally, to lower the implementation barrier, we present *Decent-DP*, a lightweight, modular software library that integrates seamlessly with existing PyTorch workflows. Decent-DP enables transparent experimentation with various topologies and the AWC communication-computation pattern. Collectively, this work bridges the gap between systems-level optimization and learning-theoretic robustness, establishing decentralized learning as a potential component for resilient distributed training systems.

Keywords

Distributed Optimization, Decentralized Learning, Data Parallelism, Straggler Mitigation, Generalization, Deep Learning Systems.

Sammanfattning

Den växande klyftan mellan beräkningskapacitet och nätverksbandbredd mellan noder i moderna datacenter utgör en fundamental utmaning för distribuerad djupinlärning. När träningskluster skalas upp så domineras systemets exekveringstid alltmer av kommunikationskostnaderna för global synkronisering (t.ex. All-Reduce). Decentraliserad träning, som ersätter globala kollektiva operationer med gles punkt-till-punkt-kommunikation, erbjuder ett teoretiskt effektivt alternativ. Trots detta har dess praktiska tillämpning hindrats av oklarheter kring effektivitet, generaliseringsegenskaper och implementeringskomplexitet.

Denna avhandling undersöker hur decentraliserad träning kan utvecklas till ett robust och effektivt alternativ till centraliserad synkronisering. Genom att begränsa kommunikationen till en gles graf av grannar minskar decentraliserade algoritmer bandbreddsanvändningen och reducerar problemen med eftersläntrare (“stragglers”) som är inneboende i globala kollektiva operationer. Trots dessa teoretiska fördelar har den praktiska användningen av decentraliserad träning hindrats av tre huvudsakliga utmaningar: oklarhet kring effektivitetsvinster, osäkerhet kring generaliseringsprestanda samt höga implementeringströsklar.

För att adressera oklarheten kring effektivitet föreslår vi en omfattande exekveringstids-modell som karakteriserar förmågan hos decentraliserade algoritmer. Vi härleder en analytisk gräns som definierar de hårdvaru- och modellregimer under vilka decentraliserad träning kan överträffa All-Reduce med betydande marginal, och validerar denna modell på GPU-kluster. Denna analys belyser relevansen av decentraliserade algoritmer för synkronisering av den “yttre loopen” i bandbredds begränsade miljöer.

För det andra hanterar vi osäkerheten kring generaliseringsförmåga genom att analysera konsensusfelets roll. Vi föreslår initialt AccumAdam, en teknisk stabiliseringsmekanism utformad för att mildra den momentumdrift som orsakas av decentralisering och för att stabilisera konvergensen. Vi skiftar sedan till ett nytt

perspektiv med DSGD-AC (Adaptive Consensus), där vi visar att konsensusfel — som ofta betraktas som skadligt brus — kan fungera som en implicit regulariseringsmekanism kopplad till kurvatur. Genom att kontrollera snarare än eliminera detta fel, kan vi få decentraliserad träning att gynna släta minima (smooth minima) och förbättra generaliseringen jämfört med centraliserade baslinjer.

Slutligen, för att sänka implementeringströskeln, presenterar vi Decent-DP, ett lättviktigt och modulärt mjukvarubibliotek som kan integreras sömlöst med befintliga PyTorch-arbetsflöden. Decent-DP möjliggör transparenta experiment med olika topologier och AWC-kommunikationsmönster. Sammanfattningsvis överbryggar detta arbete klyftan mellan optimering på systemnivå och inlärningsteoretisk robusthet, och etablerar därmed decentraliserad inlärning som en potentiell komponent för resilienta distribuerade träningsystem.

Nyckelord

Distribuerad optimering, decentraliserat lärande, dataparallellism, straggler-begränsning, generalisering, djupinlärningssystem.

Acknowledgment

It has been three years since I started my PhD journey. Reflecting on the journey leading to this work, I realize how much of a collaborative effort this has been.

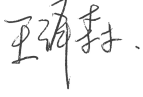
First and foremost, I would like to express my sincere gratitude to my supervisor, Prof. Mikael Johansson, for your invaluable guidance, patience, and support. Your deep insights into optimization and your ability to ask the right questions have been very helpful, and the discussion with you has always been inspiring and encouraging. I do look forward to our collaboration in the rest of my PhD journey!

I'm grateful to have Prof. Giuseppe Belgioioso as my advance reviewer and Hao-Jun Michael Shi as my opponent.

I would also like to acknowledge my great colleagues. It is my honor to work alongside brilliant minds at DCS, and thank you for your interesting and insightful discussions and support in life. Working alongside such a talented team in Mikael's group has been a true highlight of my journey. I want to extend my heartfelt thanks to every colleague — both former and current — who made the past three years so memorable. Special appreciation goes to my collaborators, Jiaojiao Zhang, Jacob Lindbäck, and Xuyang Wu. The collaborations with you were enjoyable, and I would not have gained the momentum on research in the first years without your guidance.

Finally, I would like to thank my parents for their unconditional support, love, and understanding, and I am sorry I have not been with you more. I would also like to thank my beloved wife, Yi, for her company along the journey, and for her patience and encouragement when I was down. I would not have made it here without you.

Sincerely,

A handwritten signature in black ink, appearing to be the Chinese characters '王泽森' (Wang Zesen) followed by a period.

Zesen Wang
Stockholm, February 5, 2026

List of included papers

Contributions This licentiate dissertation consists of two parts. The first part provides an overview of the research field on which I focused and a summary of my contributions to it. The second part comprises the contributions that I have made to this research field through published and submitted works.

Paper A **From promise to practice: realizing high-performance decentralized training**, Zesen Wang, Jiaojiao Zhang, Xuyang Wu, and Mikael Johansson. In The Thirteenth International Conference on Learning Representations (ICLR 2025), 2025

Paper B **DSGD-AC: controlled consensus errors improve generalization in decentralized training**, Zesen Wang and Mikael Johansson. NeurIPS 2025 Workshop on Optimization for Machine Learning (OPT-2025), 2025

Paper C **Controlled disagreement improves generalization in decentralized training**, Zesen Wang and Mikael Johansson. Under double-blind review, 2026

Other contributions by the author not included in the thesis.

Paper D **Bringing regularized optimal transport to lightspeed: a splitting method adapted for GPUs**, Jacob Lindbäck, Zesen Wang, and Mikael Johansson. In Advances in Neural Information Processing Systems, 2023

Note: I was responsible for the improvement of the GPU kernels and the GPU-related numerical experiments.

Contents

Abstract	v
Sammanfattning	vii
Acknowledgment	ix
List of included papers	xi
Contents	xiii
List of Figures	xv
List of Tables	xvii
Listings	xviii
1 Introduction	1
1.1 Background and motivation	1
1.2 Challenges	5
1.3 Contributions and Thesis Structure	6
2 Background	11
2.1 Hardware topology of HPC clusters: the case of Alvis	11
2.2 Centralized distributed training frameworks	12
2.3 Decentralized optimization algorithms	14
2.4 Sharpness and generalization in deep learning	18
3 Preliminaries and Problem Setup	23
3.1 Notations	23
3.2 Problem setup	23
4 Efficient decentralized learning	27
4.1 Overview	27
4.2 Key factors for efficient multi-node decentralized training	27
4.3 Runtime model for predicting speedup	32
4.4 Numerical experiments	37
4.5 Summary	39
5 Robust Decentralized Training	41
5.1 Overview	41
5.2 Alleviate the gap caused by decentralization	42

CONTENTS

5.3	Exploit consensus errors as curvature regularizer	46
5.4	Summary	65
6	Design and implementation of Decent-DP	67
6.1	Motivation and design goal	67
6.2	Implementation details	67
6.3	Usage and integration	68
6.4	Profiling results	72
6.5	Summary	74
7	Conclusions and future work	75
7.1	Concluding remarks	75
7.2	Limitations and future research directions	75
	Bibliography	77

List of Figures

1.1.1	The exponential growth trend in the number of parameters of notable AI models from 2010. The data source is from epoch.ai by Oct. 2025 [7].	1
1.3.2	Thesis Organization.	9
2.4.1	A conceptual plot of flat minimum and sharp minimum.	19
4.2.1	Example timeline of All-Reduce training. F : forward pass, B_i : backward pass for bucket i , C_i : gradient communication for bucket i , and U : optimizer update. The arrows represent the dependencies.	28
4.2.2	Example timeline of the adapt-while-communicate decentralized training. F : forward pass, B_i : backward pass for bucket i , C_i : parameter communication for bucket i , and U_i : optimizer update for bucket i . The arrows represent the dependencies.	28
4.2.3	The communication operations in each iteration for the one-peer exponential topology. The circles are the GPUs, the dashed lines are the boundaries of the nodes, the lines connecting the circles are the communication operations, and the lines in red are the communication operations that span across multiple nodes.	30
4.2.4	The communication operations in each iteration for the alternating exponential ring topology. The circles are the GPUs, the dashed lines are the boundaries of the nodes, the lines connecting the circles are the communication operations, and the lines in red are the communication operations that span across multiple nodes.	31
4.2.5	Convergence comparison of various topologies with 16 workers. The initial values of the workers are randomly sampled from a normal distribution.	31
4.2.6	Distributions of the normalized computation time in two tasks. Image classification: ResNet50 on ImageNet. Neural machine translation: Transformer (base) on WMT14 English-German.	32
4.3.7	Runtime model and actual runtime in the experiment of training Transformer (base) on WMT14 English-to-German.	34
4.3.8	Simulation by the runtime model. In the simulation, $N = 8$ (8 workers), $b = 4$ (4 buckets, from ResNet-50 experiments), $\theta = 0.05$ (time taken by updating one bucket is 0.05 unit time), and the communication topology of decentralized training used for determining the neighbors is Complete topology.	35
4.4.9	The results show the strong scaling performance (fixed global batch size, 25K tokens) of All-Reduce, 1-OSGP [23] and our implementations by reporting the per-iteration runtime with various numbers of workers and network settings. Here, Exp and AER stand for one-peer exponential graph and alternating exponential ring, respectively.	37

LIST OF FIGURES

4.4.10 Training a transformer (base) model on the English-to-German translation task on four 4x4A40 nodes with 25Gbps Ethernet connections. The error bands of $\pm 2\sigma$ are based on three runs. The red dashed line on the left is the baseline (27.3 BLEU score) from [31]. 37

4.4.11 The curves of train and validation losses where the x-axis is training time. AR: All-Reduce training. Decent: Decentralized training with AccumAdamW. 39

5.3.1 Decentralized training of WRN28-10 on CIFAR-10 (3 random runs for each algorithm) with 8 workers, and the communication topology is the one-peer ring topology. **Left:** Learning rate schedule (same for both algorithms). **Right:** Average norm of consensus errors evaluated at the end of every epoch ($\frac{1}{N} \sum_{i=1}^N \|x_i^{(eT)} - \bar{x}^{(eT)}\|$). p is set to 3 for DSGD-AC. 47

5.3.2 **Left:** Losses on the whole training dataset at local workers and global average. The losses are evaluated every 10 epochs. **Right:** Training loss at epoch 180 along: (1) worker i : lines connecting global average and worker i , (2) gradient: the line that aligns with the full-batch gradient at the global average and crosses the global average, and (3) random: 500 lines that cross the global average and follow random directions generated as in [61]. The x -axis means the directional magnitude of the perturbation along these directions. The red dots represent the losses at the local models. The losses are computed on $\sim 1/4$ of the training dataset due to computation complexity. 52

5.3.3 WRN28-10 on CIFAR-10. **Left:** Test accuracy on test set. For decentralized training, the accuracy is evaluated on the global average model. **Right:** Training and test losses. 62

5.3.4 WRN28-10 on CIFAR-100. **Left:** Test accuracy on test set. For decentralized training, the accuracy is evaluated on the global average model. **Right:** Training and test losses. 62

5.3.5 WRN16-8 on CIFAR-10. **Left:** Test accuracy on test set. For decentralized training, the accuracy is evaluated on the global average model. **Right:** Training and test losses. 62

5.3.6 WRN16-8 on CIFAR-100. **Left:** Test accuracy on test set. For decentralized training, the accuracy is evaluated on the global average model. **Right:** Training and test losses. 63

5.3.7 DSGD(-AC) on WRN28-10 on CIFAR-10 with varying p . **Left:** Test accuracy on test set. For decentralized training, the accuracy is evaluated on the global average model. **Right:** Training and test losses. 63

5.3.8 Average norm of consensus errors over epochs with varying p 64

5.3.9 Transformer (big) on WMT14 English-to-German. **Left:** Losses on training set. **Right:** BLEU scores on the test set. 64

6.4.1 Profiling results for PyTorch DDP. Setup: $2 \times 4 \times A40$ GPU nodes interconnected by 25Gbps Ethernet. 73

6.4.2 Profiling results for Decent-DP. Setup: $2 \times 4 \times A40$ GPU nodes interconnected by 25Gbps Ethernet. Topology: Alternating exponential ring. 73

List of Tables

1.1.1	Comparison of FP16 tensor performance (commonly used metric for ML workload performance), NVLink bandwidth, and the ratio of NVLink bandwidth to tensor performance of NVIDIA GPUs. A smaller ratio indicates that compute performance has grown faster than interconnect bandwidth. Data is vendor-repeated peak values, and the effective training performance depends on kernel efficiency and memory bandwidth.	4
2.1.1	Communication bandwidth hierarchy for various GPU node types in Alvis.	12
2.4.2	Summary and comparison of SAM [59], Adaptive SAM [60] and LPF-SGD [61].	20
4.2.1	Communication time of performing averaging operations for 8192 iterations over three 25MB FP32 tensors. Setup 1: 16 A100 GPUs on 4 nodes inter-connected by 100Gbps Infiniband. Setup 2: 16 A40 GPUs on 4 nodes inter-connected by 25Gbps Ethernet. Here, Complete means global averaging.	30
4.4.2	Results of speedup of training ResNet-50 on ImageNet dataset for the image classification task. The topology is complete graph for decentralized training.	38
5.2.1	Neural Machine Translation on WMT14 [76]: The results show the generalization performance of training transformer (base) on English-to-German and transformer (big) on English-to-French. The last checkpoint of each method is evaluated by BLEU score [68] and METEOR [69] the corresponding newtest2014 testset. The number of workers is 16 and the number of workers per node is 4 for all experiments. The error bands with $\pm 2\sigma$ based on 3 runs are reported.	46
5.2.2	Image Classification on ImageNet-1K [78]: The results are show the generalization of training ResNet-50 on ImageNet-1K for image classification. Top-1 and Top-5 accuracies and averaged training loss in the last epoch are reported, and the accuracies are evaluated on the ImageNet validation set. The number of workers is 32 and the number of workers per node is 8 for all experiments.	46
5.3.3	Algorithm comparison on image classification. We use 8 workers and one-peer ring topology for decentralized methods. The best in each experiment setup is bold , and the second best is <u>underlined</u>	61
5.3.4	Sensitivity analysis of parameter p in the WRN28-10 on CIFAR10 experiment. The best value is bold , and the second best is <u>underlined</u>	64
5.3.5	Performance comparison of DAdam, Adam, and DAdam-AC on neural machine translation with the transformer model.	65

Listings

6.1	Install Decent-DP	68
6.2	Model wrapper in PyTorch DDP	69
6.3	Model wrapper in Decent-DP	69
6.4	Training example with PyTorch DDP	70
6.5	Training example with Decent-DP	70
6.6	One-peer ring topology defined in Decent-DP	70
6.7	Training example with adaptive consensus in Decent-DP	71

1 Introduction

1.1 Background and motivation

In recent years, the deep learning landscape has undergone a significant expansion. The rapid scaling of model architectures, from early convolutional networks and small transformer encoders to today's large language models (LLMs), large vision-transformer systems, and diffusion models for image and video generation, has become the norm rather than the exception. Figure 1.1.1 demonstrates an exponential growth in the model size from 2010. These modern models often comprise billions, or in some cases hundreds of billions, of parameters and are trained on petabytes of data [5, 6].

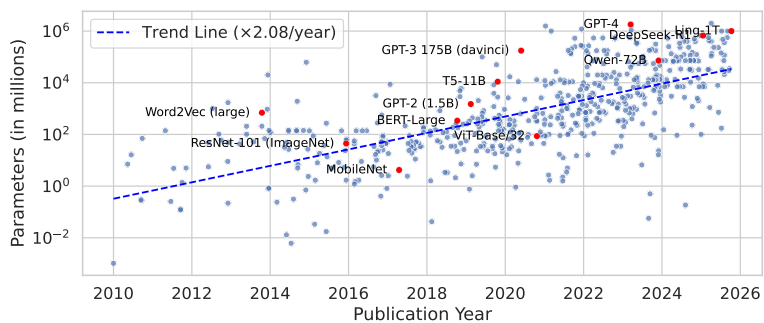


Figure 1.1.1: The exponential growth trend in the number of parameters of notable AI models from 2010. The data source is from epoch.ai by Oct. 2025 [7].

This unprecedented scaling has pushed the limits of hardware capacity. The computational demands of training these models far exceed what a single GPU can

provide. Consequently, distributed training that splits computation across multiple devices has become not merely an optimization but a necessity for modern AI workloads.

1.1.1 Dimensions of parallelism

Distributed training encompasses multiple dimensions of parallelism:

- Data parallelism (DP) replicates the model on each worker and partitions the input data batch, followed by gradient aggregation. This strategy is conceptually simple and scales near-linearly with the batch size, making it the dominant method in most large-scale deep-learning systems [8].
- Pipeline parallelism (PP) divides the model into sequential stages distributed across GPUs, allowing different microbatches to flow through the pipeline concurrently [9]. This reduces the memory requirements on a single device but introduces pipeline bubbles and scheduling complexities.
- Tensor parallelism (TP) splits individual neural-network layers, typically matrix multiplications, across devices [10, 11]. This is essential when even a single layer exceeds the memory capacity of one GPU, as in billion-parameter-scale large language models.
- Fully sharded data parallelism (FSDP) [12] / zero redundancy optimizer (ZeRO) [13] partitions the optimizer states, gradients, and model parameters across data-parallel workers instead of replicating the full model on every device. This approach drastically reduces memory redundancy, enabling the training of models with trillions of parameters. However, this memory efficiency comes at the cost of increased communication overhead for scattering and gathering parameters across workers during every forward and backward pass.

This thesis focuses specifically on the data parallelism to isolate and understand the fundamental trade-offs among communication topology, runtime efficiency, and generalization capability. It provides the necessary algorithmic foundations that can subsequently be adapted to the more complex, hybrid-parallel architectures of frontier-scale systems.

In large-scale systems such as Megatron-LM [10], DeepSpeed [5], and PaLM [14], distributed training is facilitated by hybrid parallelism, combining data, tensor, and pipeline strategies to balance memory usage and throughput. While data parallelism may not always dominate the total wall-clock time compared to the more frequent communication operations of tensor parallelism or FSDP, it is still critical because it relies heavily on inter-node communication. Unlike tensor parallelism, which typically exploits ultra-fast intra-node interconnects (e.g., NVLink), data parallel synchronization must traverse the slower inter-node networks (e.g., InfiniBand or Ethernet) where bandwidth is the most limited. Consequently, optimizing this specific communication dimension remains one of the core bottlenecks for improving scaling efficiency. Moreover, DP is considered to be orthogonal and composable with TP, PP and FSDP.

1.1.2 Data parallelism

Two mainstream centralized paradigms dominate distributed deep-learning practice:

- Parameter server (PS) architectures maintain one or several dedicated nodes to aggregate and distribute parameter updates from workers [15]. This design offers flexibility, particularly for asynchronous updates, but suffers from server bottlenecks and single node of failure in server. The architecture also supports asynchronous training but introduces randomly stale gradients.
- All-Reduce-based methods perform collective communication among all workers to average gradients at each iteration. Implementations such as ring All-Reduce or hierarchical tree reduction efficiently utilize bandwidth and are now the default in frameworks like PyTorch’s DistributedDataParallel [16].

Both paradigms are robust and mature in some scenarios, but the advantages of All-Reduce-based data parallelism in alleviating workload imbalance made it more prevailing and adopted by the community as the mainstream in recent years. However, their limitations become apparent at a large scale or with sub-optimal hardware:

- Communication latency and the total traffic volume across the network grow linearly with the number of workers for PS and ring All-Reduce.
- Global synchronization causes idle time, especially under hardware heterogeneity or straggler effects.
- With suboptimal interconnection, communication time may dominate the iteration time in centralized methods.

1.1.3 Communication bottleneck in ML hardware

Despite the impressive progress in compute performance, communication has increasingly emerged as the dominant bottleneck in large-scale machine learning systems. Table 1.1.1 lists the computational performance and the inter-connection performance of Nvidia GPUs that are widely used in data centers [17]. As shown in Table 1.1.1, the growth rate of GPU tensor-core performance has significantly outpaced the improvement in interconnect bandwidth. From the NVIDIA Tesla V100 to the Blackwell generation, floating-point throughput has increased by $20\times$, whereas the NVLink bandwidth has only improved by $6\times$. Consequently, the ratio between compute capability and communication bandwidth continues to widen, indicating that communication links evolve at a slower pace than computation.

This imbalance implies that as GPUs become more powerful, the relative cost of synchronizing gradients or parameters across devices becomes increasingly pronounced. In large-scale All-Reduce or parameter-server setups, this results in higher communication overhead per training iteration, leading to sublinear scaling efficiency as the cluster size grows. Even when using the latest flagship GPUs, which can reduce the total number of required devices, scaling efficiency is still limited because network interconnects—both intra-node (e.g., NVLink, NVSwitch) and inter-node (e.g., InfiniBand, Ethernet)—do not scale proportionally with compute

Model	FP16 Performance	NVLink Bandwidth	Bandwidth / Computation Ratio
Tesla V100	125 TFLOPS	300 GB/s	2.40
A100	312 TFLOPS	600 GB/s	1.92
H100	989.4 TFLOPS	900 GB/s	0.91
Blackwell	2500 TFLOPS	1800 GB/s	0.72

Table 1.1.1: Comparison of FP16 tensor performance (commonly used metric for ML workload performance), NVLink bandwidth, and the ratio of NVLink bandwidth to tensor performance of NVIDIA GPUs. A smaller ratio indicates that compute performance has grown faster than interconnect bandwidth. Data is vendor-repeated peak values, and the effective training performance depends on kernel efficiency and memory bandwidth.

throughput. In other words, while fewer, faster GPUs can mitigate some communication needs, they cannot eliminate the fundamental disparity between compute and communication scaling.

The practical consequence is that communication often dominates the iteration time in distributed training workloads, especially when training models with a large number of parameters or small per-GPU batch sizes. This growing mismatch has motivated the exploration of strategies that reduce synchronization frequency, limit global communication, or restructure communication patterns altogether. This growing mismatch motivates the search for paradigms that reduce or localize synchronization, which is the central idea behind decentralized training.

1.1.4 Decentralized training as an emerging alternative

The idea of decentralized optimization has its roots in the broader field of distributed consensus algorithms, originally studied in control theory and sensor networks during the early 2000s [18, 19, 20]. Its modern incarnation in machine learning dates back to the context of federated learning [21], where the goal was to train a global model across multiple devices or institutions without a central coordinator collecting raw data. These early systems demonstrated that local computation combined with model averaging could achieve a degraded but reasonable accuracy while preserving the privacy of the local data of end users.

In recent years, this idea has migrated from cross-device federated scenarios to high-performance data-center environments, where communication efficiency and scalability, rather than privacy, are the primary concerns [22, 23, 24]. The motivation is straightforward: as models and datasets grow, even the most optimized centralized approaches, such as All-Reduce collectives or parameter server architectures, struggle with synchronization delays, communication bottlenecks, and stragglers. Decentralized training offers a promising alternative by removing any single point of coordination and global synchronization, and replacing it with a fully peer-to-peer (or small-group) communication model.

In decentralized training, every worker maintains its own local copy of the model parameters. During each iteration, each worker computes gradients on its local data and then exchanges parameters only with a subset of its peers, as determined by a communication topology (a graph connecting the nodes).

A representative decentralized algorithm is Decentralized Stochastic Gradient Descent (D-PSGD) [22]. The update rule takes the form:

$$x_i^{(t+1)} = \underbrace{\sum_{j \in \mathcal{N}(i)} W_{ij} x_j^{(t)}}_{\text{decentralized communication}} - \underbrace{\alpha \nabla f_i(x_i^{(t)})}_{\text{local update}} \quad (1.1.1)$$

where $x_i^{(t)}$ denotes the local model of worker i at iteration t , α denotes the learning rate, and $\mathcal{N}(i)$ denotes the neighbors of worker i .

Unlike All-Reduce training, which requires a global synchronization barrier after every gradient computation, D-PSGD allows communication and local computation to proceed in parallel. Each worker can perform the next forward and backward passes while exchanging model updates with its neighbors, effectively overlapping computation and communication. This overlap reduces idle time and improves GPU utilization, especially in bandwidth-constrained settings or when inter-node latency is high [22, 23]. Furthermore, because each node communicates with only a few peers, the per-node communication cost scales as $\mathcal{O}(1)$ instead of $\mathcal{O}(n)$, leading to better scalability for large clusters.

The decentralized approach also offers architectural flexibility: communication topologies can be customized to match physical network layouts, such as ring, torus, or exponential graphs, to reduce hop distance and latency. Additionally, the absence of a central node or global synchronization eliminates a single point of failure, improving fault tolerance and robustness to stragglers.

1.2 Challenges

Given these potential advantages in efficiency, one might expect decentralized training to be the default approach in distributed deep learning. However, this is not yet the case. Despite its theoretical efficiency and resilience, decentralized training remains far less adopted than centralized paradigms like All-Reduce or parameter server architectures. The gap arises not from lack of promise, but from several practical and theoretical challenges that complicate its adoption and understanding. These challenges can be broadly categorized into three dimensions:

Efficiency ambiguity Although decentralized training offers compelling theoretical advantages (for example, sparse communication patterns and better overlap of computation and communication), its real-world runtime benefits are not guaranteed. One of the main obstacles is the complex design space underlying decentralized systems, involving communications topology, computation-communication overlap patterns, optimization algorithm variants, hardware interconnects, and more. In practice, the mismatch of any of these dimensions can erase the expected speedups. Without a unified runtime model to navigate these interacting factors, practitioners cannot reliably predict when decentralized training will deliver actual wall-clock speedups, leading to hesitant adoption.

Uncertain generalization The second challenge concerns how decentralization affects optimization and generalization. In many decentralized training works [22, 23, 25], decentralized training is considered as a defective version of its centralized counterpart in terms of convergence and generalization because of consensus errors. The canonical view, rooted in convex analysis, treats consensus errors as harmful noise that slows convergence and degrades accuracy. This is largely true for convex objectives, where maintaining close agreement among nodes ensures faster descent. However, for non-convex neural networks, this assumption may not hold: training loss and test performance are not always positively correlated, and moderate disagreement among local models might even help escape sharp minima. Some initial attempts have been done to formalize this potential benefit, showing that consensus noise can act like a regularization term similar to average-direction sharpness-aware minimization [26]. Still, the intrinsic properties of the consensus error have not been studied in depth, and there are no algorithms that actively controls the consensus errors to exploit the resulting regularization. Whether consensus errors are ultimately beneficial or detrimental remains an open and under-explored question.

Implementation barrier Finally, decentralized training remains difficult to adopt in practice due to implementation barriers. Existing frameworks such as BlueFog [27] and Bagua [28] are efficient distributed/decentralized training frameworks, but they depend on external communication libraries, and lack in flexible high-level modular interfaces for algorithmic research. Their maintenance activity has also declined, making reproducibility and extensibility difficult. To bridge this gap, a simple, lightweight, research-friendly, and, at the same time, efficient framework is needed to enable fair runtime comparison and rapid prototyping.

1.3 Contributions and Thesis Structure

This thesis includes the following interconnected contributions:

- Analyze the key factors in multi-node decentralized training that enable speedup over All-Reduce training, and provide a predictive model quantifying the mentioned factors and identifying regimes where decentralized training achieves maximal speedups over centralized methods.
- Analyze the convergence of the decentralized variant of Adam optimizer, and propose a novel variant *AccumAdam* facilitating the training of Transformer-based models with comparable generalization to All-Reduce training.
- Propose *DSGD-AC*, a novel variant of *D-PSGD* that can achieve superior generalization than centralized SGD, which provides the empirical base for analyzing the intrinsic property of consensus errors.
- Implement *Decent-DP*, an open-source, light-weight, efficient, modular PyTorch extension that embodies insights from both the efficiency and generalization studies and bridges algorithmic research and practical deployment.

Together, these contributions aim to improve our understanding of *when decentralized training is faster, how it can generalize better, and how to implement it effectively*.

The thesis covers the following submissions:

- Paper A: **Zesen Wang**, Jiaojiao Zhang, Xuyang Wu, and Mikael Johansson. *From Promise to Practice: Realizing High-performance Decentralized Training*. The Thirteenth International Conference on Learning Representations (ICLR 2025).
- Paper B: **Zesen Wang**, and Mikael Johansson. *DSGD-AC: controlled consensus errors improve generalization in decentralized training*. OPT2025: 17th Annual Workshop on Optimization for Machine Learning (held in conjunction with NeurIPS 2025).
- Paper C: **Zesen Wang**, and Mikael Johansson. *Controlled disagreement improves generalization in decentralized training*. Under double-blind review.

Note: Paper C is an updated version of Paper B with improved theoretical analysis and additional numerical experiments.

Note on my own contributions For all the papers (A, B, and C), I am the first author and the primary contributors. In Paper A, the co-authors contributed to the ideas and refinement of the convergence proof and insight discussions on improving the paper. In Paper B and Paper C, the co-author contributed to part of the theoretical results.

The main body of the thesis is organized into five chapters. The first covers background and related works, the second covers notations, preliminaries and the scope of the thesis, the remaining three chapters present how decentralized training can be implemented efficiently with superior generalization. More specifically, the thesis is organized as follows:

- **Chapter 2 - Background:** The chapter introduces the background and related works on distributed and decentralized training paradigms and algorithms, and generalization of deep neural networks.
- **Chapter 3 - Preliminaries and problem setup:** The chapter introduces the notations that are used throughout the thesis, the related metrics for evaluation, and the problem setup that the thesis focuses on.
- **Chapter 4 - Efficient decentralized training:** The chapter addresses how, when, and how much decentralized training can achieve speedup over All-Reduce training with a predictive runtime model quantifying all analyzed aspects. It establishes the theoretical and empirical foundation for understanding the runtime efficiency of decentralized algorithms under different system and network configurations. The chapter is mainly based on Paper A.
- **Chapter 5 - Robust decentralized training:** The chapter investigates how decentralization influences optimization and generalization performance. It

first studies the convergence properties of decentralized adaptive methods, introducing the *AccumAdam* optimizer for stable training of Transformer-based architectures. Then, it explores the role of consensus errors in generalization, proposing *DSGD-AC*, a controlled-consensus variant of decentralized SGD that leverages disagreement as an implicit regularizer to improve test performance. The chapter is mainly based on Paper B and Paper C.

- **Chapter 6 - Design and implementation of *Decent-DP*:** This chapter presents *Decent-DP* (stands for decentralized data parallelism), an open-source, lightweight, and modular PyTorch extension for decentralized training. It details the software design principles, communication backends, and modular components that enable flexible experimentation with various topologies, optimizers, and algorithms. The chapter also includes benchmarking results and demonstrates how insights from the theoretical and empirical studies are incorporated into the framework design. The chapter is mainly based on Paper A, but all papers use or extend *Decent-DP* for numerical experiments.
- **Chapter 7 - Conclusion and future work:** The final chapter summarizes the main findings and contributions of the thesis. It highlights how the proposed models, algorithms, and tools advance the understanding and practical usability of decentralized training. The chapter concludes with a discussion on open challenges and outlines promising future research directions in efficient, robust, and adaptive decentralized learning.

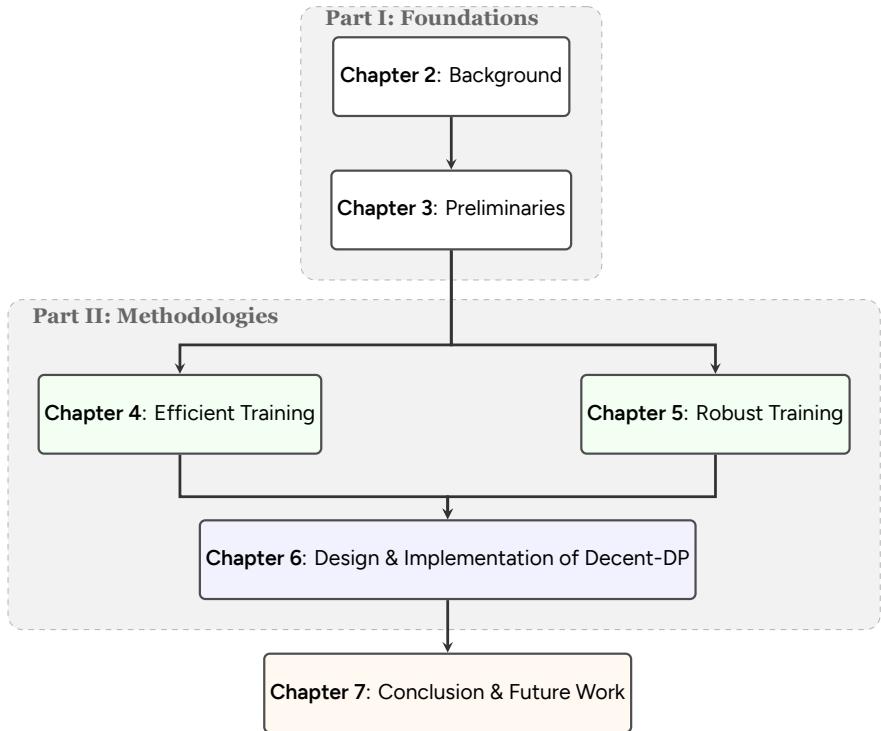


Figure 1.3.2: Thesis Organization.

2 Background

2.1 Hardware topology of HPC clusters: the case of Alvis

To better understand the system-level motivation for decentralized training, it is important to examine the communication characteristics of modern high-performance computing (HPC) clusters used for large-scale AI workloads. This section describes the typical hardware topology of such clusters, using the *Alvis* system at Chalmers University of Technology as a representative example, while highlighting recent architectural trends in large-scale GPU clusters. The *Alvis* system is the testbench for all numerical experiments included in the thesis.

Overview of the Alvis system *Alvis* is part of the Swedish National Academic Infrastructure for Supercomputing (NAISS) and is operated by the Chalmers Centre for Computational Science and Engineering (C3SE). It is designed primarily for AI, machine learning, and data-intensive workloads, and provides a diverse set of GPU-accelerated compute nodes interconnected through high-bandwidth networks. *Alvis* features a heterogeneous composition of nodes equipped with various generations of GPUs (including NVIDIA A100, A40, V100, and T4 accelerators), CPU architectures, and memory capacities. GPU nodes contain four to eight GPUs per node for compute-intensive deep-learning tasks, while others are optimized for lightweight inference or visualization workloads.

Communication link heterogeneity Like most modern GPU clusters, *Alvis* exhibits a clear hierarchy in communication bandwidth and latency:

- Within a GPU, data movement occurs through on-chip interconnects between compute cores, caches, and high-bandwidth memory (HBM). The intra-GPU memory bandwidth—often exceeding 1 TB/s on recent architectures—enables extremely fast access to model parameters and activation tensors. However,

despite the high throughput, performance can still be limited by memory access patterns and kernel fusion efficiency, making optimal intra-GPU data locality crucial for full utilization.

- Within a node, GPUs are connected by high-bandwidth links such as NVLink or PCIe, enabling nearly uniform and low-latency communication among GPUs. This makes intra-node collectives (e.g., local All-Reduce) highly efficient and typically negligible in training overhead.
- Across nodes, communication relies on InfiniBand or high-speed Ethernet interconnects through switches. Although these links are optimized for RDMA and collective operations, their bandwidth is typically one or two orders of magnitude lower than NVLink or PCIe. Consequently, cross-node synchronization (as required in every iteration in All-Reduce or parameter-server training) often dominates iteration time once training scales out.

The statistics of some typical setups in *Alvis* are listed in Table 2.1.1. This hierarchy implies that the cost of communication grows steeply with distance: intra-GPU \ll intra-node \ll inter-node. Thus, algorithms that minimize cross-node synchronization are essential for scalability.

Link Type \ Node	8×T4 Node	4×A40 Node	4×A100 Node
Intra-GPU	300 GB/s	696 GB/s	1935 GB/s
Intra-Node	32 GB/s (PCIe)	64 GB/s (PCIe)	600 GB/s (NVLink)
Inter-Node	12.5 GB/s (InfiniBand)	3.125 GB/s (Ethernet)	12.5 GB/s (InfiniBand)

Table 2.1.1: Communication bandwidth hierarchy for various GPU node types in Alvis.

2.2 Centralized distributed training frameworks

Centralized distributed training frameworks provide the foundation for nearly all large-scale deep learning systems used in both research and industry. They are typically built upon collective communication primitives and rely on tightly synchronized gradient exchange to ensure model consistency across all participating workers. This section reviews the key implementation mechanisms that make centralized training efficient in practice and highlights their inherent limitations that motivate decentralized alternatives.

2.2.1 Gradient bucketing and overlap strategies

A key optimization in data parallelism is the *gradient bucketing* mechanism, introduced to reduce communication latency and overlap computation with communication. The optimization is widely applied in centralized distributed training frameworks, such as PyTorch Distributed Data Parallel (DDP), the mirror strategy in TensorFlow, Horovod [29], and BytePS [30].

In *gradient bucketing*, instead of waiting for the entire backward pass to finish before initiating gradient aggregation, modern frameworks divide model parameters into

multiple *buckets*, each containing a subset of gradients. Once the gradients in one bucket are computed, they are immediately communicated across workers using an All-Reduce operation, while subsequent layers continue the backward computation in parallel.

This design enables partial communication-computation overlap: the GPU can perform backpropagation on deeper layers while it concurrently transmits gradients of earlier layers. However, the degree of overlap heavily depends on the bucket size and network bandwidth. Larger buckets yield higher bandwidth utilization but reduce overlap opportunities, while smaller buckets increase latency overhead. As a result, frameworks such as `PyTorch DistributedDataParallel (DDP)` tune bucket sizes dynamically or heuristically to balance these effects [16].

2.2.2 Communication backends and collective primitives

Modern distributed frameworks rely on high-performance communication libraries that implement collective operations efficiently across heterogeneous interconnects. For Nvidia GPUs, NCCL (NVIDIA Collective Communications Library) is the standard for GPU collectives, offering highly optimized ring and tree-based All-Reduce algorithms that fully exploit NVLink and PCIe topologies. NCCL automatically discovers the GPU interconnection graph and chooses an optimal reduction pattern to minimize hop latency and bandwidth contention.

In PyTorch, the distributed framework also integrates the following backends:

- **MPI (Message Passing Interface):** A general-purpose communication standard widely adopted in HPC. It enables cross-vendor compatibility but often at the cost of higher communication overhead compared to NCCL.
- **Gloo:** A CPU-based backend designed for portability and ease of use. It supports both TCP and InfiniBand transports and serves as a fallback option in PyTorch when GPUs are unavailable.

Collective primitives form the core of gradient synchronization. Among them, All-Reduce is the dominant operation, responsible for aggregating gradients across all workers every iteration.

2.2.3 Sensitivity to stragglers

Stragglers caused by imbalance workloads In the training of language models, one way to preprocess the input data is to group samples with similar lengths [31], but this may introduce the straggler effect because of non-uniform batches. Increasing the number of local mini-batches [32] or dropping samples on stragglers [33] mitigate the problem but may worsen the generalization performance by altering the effective batch size [34]. In pre-training LLMs, it is common to pack the sequences from different documents [35] to create uniform batches. However, the sequence pack technique has a negative influence on the training quality [36]. Another possibility is to pad sequences to a fixed length [37], but this leads to lower resource utilization. Moreover, the methods are not compatible with tasks like video processing and other seq-to-seq tasks.

Stragglers caused by system noise Recent empirical studies on large-scale clusters confirm that stragglers are not merely transient anomalies but a prevalent bottleneck. For instance, production traces [38] reveal that over 40% of large training jobs suffer significant slowdowns due to fail-slow hardware. Furthermore, in hybrid parallelism settings, these delays are amplified: a single straggler can trigger a “domino effect” of pipeline bubbles or continuously stall an entire tensor-parallel group, rendering the straggler effect more severe than in pure data-parallel setups [39].

The straggler problem poses a major challenge on synchronous centralized training. In the All-Reduce paradigm, the global gradient aggregation cannot complete until the slowest worker has finished its computation [16]. While the straggler problem is hard to be avoided, the improvement on conventional data parallelism is necessary.

2.3 Decentralized optimization algorithms

As the scale of data and model parameters continues to grow, fully synchronized distributed training becomes increasingly difficult to sustain. Decentralized optimization algorithms offer an alternative paradigm that removes the need for a central coordinator or a global synchronization barrier. Instead, multiple workers collaborate through local information exchange, collectively optimizing a shared objective in a decentralized fashion. This section briefly introduces the foundations of decentralized optimization, outlines representative algorithms, and discusses their applications to deep learning.

2.3.1 Conceptual foundations

The fundamental goal of decentralized optimization is to minimize a global objective that can be decomposed across n workers:

$$\begin{aligned} & \underset{\{x_1, \dots, x_n\} \in \mathbb{R}^{n \times d}}{\text{minimize}} && \frac{1}{n} \sum_{i=1}^n f_i(x_i), \\ & \text{subject to} && x_1 = x_2 = \dots = x_n \end{aligned} \quad (2.3.1)$$

where $f_i(x_i)$ denotes the local loss function associated with data stored on worker i and the local model x_i . Each worker maintains its own local model x_i and can communicate only with a subset of other workers, as defined by a communication graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, where \mathcal{V} represents the set of workers and \mathcal{E} the set of communication links.

2.3.2 Decentralized stochastic gradient descent

The most widely studied family of decentralized optimization algorithms are gradient-based methods. At each iteration t , worker i updates its local model as:

$$x_i^{(t+1)} = \sum_{j \in \mathcal{N}(i)} W_{ij} x_j^{(t)} - \alpha_t \nabla f_i(x_i^{(t)}), \quad (2.3.2)$$

where α_t denotes the learning rate, W is a non-negative and doubly stochastic matrix, and $\mathcal{N}(i)$ denotes the neighbors of worker i . The first term performs local averaging across neighboring workers to promote consensus, while the second term takes a stochastic gradient step using local data. This algorithm, known as *Decentralized Parallel Stochastic Gradient Descent (D-PSGD)* [22] or DSGD, generalizes standard SGD to a networked setting.

DSGD can be heuristically interpreted as approximately minimizing a penalized objective with a quadratic regularizer that is converted from the consensus constraint and weighted by $1/\alpha$,

$$J(x_1, \dots, x_n) = \sum_{i=1}^n f_i(x_i) + \frac{1}{\alpha} \sum_{i,j} W_{ij} \|x_i - x_j\|_2^2 \quad (2.3.3)$$

2.3.3 Decentralized communication topology

A central concept in decentralized methods is the *mixing matrix* $W \in \mathbb{R}^{n \times n}$, which encodes the communication topology. The matrix satisfies:

- $W_{ij} \in [0, 1]$ for all $(i, j) \in \mathcal{V} \times \mathcal{V}$;
- $W_{ij} > 0$ if and only if $(i, j) \in \mathcal{E}$, i.e., worker i communicates with j ;
- $W\mathbf{1} = \mathbf{1}$ and $\mathbf{1}^T W = \mathbf{1}^T$, ensuring *doubly stochasticity*.

Common choices of mixing matrix W include:

- **Ring:** Each worker communicate with its two adjacent neighbors.

$$W_{ij} = \begin{cases} \frac{1}{3} & i \in \{j-1, j, j+1\} \pmod{n} \\ 0 & \text{otherwise} \end{cases}$$

- **One-peer ring:** A time-varying version of the ring topology, where worker communicates with its two adjacent neighbors in turn.

$$W_{ij}^{(t)} = \begin{cases} \frac{1}{2} & i \in \{j-1, j\} \pmod{n} \text{ and } i+t = 0 \pmod{n} \\ \frac{1}{2} & i \in \{j+1, j\} \pmod{n} \text{ and } i+t = 1 \pmod{n} \\ 0 & \text{otherwise} \end{cases}$$

- **Hypercube:** Suppose that a factorization of n is $n = \prod_{k=1}^K d_k$ ($d_k \in \mathbb{N}^+$ and $d_k > 1$), and assume each worker index $i \in \{1, \dots, n\}$ is uniquely mapped to a multi-index in the hypercube as

$$i \leftrightarrow (i_1, \dots, i_K), i_k \in \{1, \dots, d_k\}$$

then, in each iteration, the worker communicates with workers that share the same index along one of the dimensions. For $t = Kt_0 + k$, the W matrix is

$$W_{ij}^{(Kt_0+k)} = \begin{cases} \frac{1}{d_k} & i_l = j_l \forall k \neq l \\ 0 & \text{otherwise} \end{cases}$$

One-peer exponential graph [40] is an special case of the hypercube topology where n is a power of 2 and $d_1 = \dots = d_K = 2$.

- Random or dynamic graphs: Edges change over time to balance communication load, improve robustness to link failures, or in the case that the network connectivity is not stable across time.
- Complete (fully-connected): All the workers do a global average. In this case,

$$W_{ij} = \frac{1}{n}$$

Average consensus problem Consensus (in particular, average consensus) is commonly used as a benchmark for evaluating the convergence performance of a communication topology in distributed and multi-agent systems. Assume a random vector $Y \in \mathbb{R}^{n \times 1}$ and its mean vector $\bar{Y} = [\frac{1}{n} \sum_i Y_i, \frac{1}{n} \sum_i Y_i, \dots]^\top$, the number of iteration taken to achieve ϵ -consensus by iteratively applying the mixing matrix on the vector typically indicates the convergence performance of a mixing matrix. To be specific, minimize the mean-square error with less iterations,

$$\text{MSE} = \|W^k Y - \bar{Y}\|_2^2 \quad (2.3.4)$$

Without further assumption on the vector Y , the best upper we can have on MSE after one iteration is

$$\|WY - \bar{Y}\|_2^2 = \|(W - W^\infty)(Y - \bar{Y})\|_2^2 \leq |\lambda_2(W)|^2 \|Y - \bar{Y}\|_2^2 \quad (2.3.5)$$

where $\lambda_2(W)$ is the eigenvalue with second largest norm among all eigenvalues of W . The *spectral gap* is then defined as $\rho = 1 - |\lambda_2(W)|$ which quantifies the convergence performance of the mixing matrix. For time-varying topologies, where W is not fixed, it can be generalizable by considering the whole loop of the topologies as one "large" step (for example, K steps in hypercube topology).

It follows directly that the consensus can be achieved in one step if using the complete topology (with $\lambda_2(W) = 0$), but, in practice, the design of the topology usually reflects the system constraints or the communication costs. In the literature of decentralized optimization, the communication cost is usually simply assumed to be uniform among all pairs of workers and be proportional to the number of neighbors.

Following the assumption, one-peer exponential topology [40] is one of the best topology if restricting the number of neighbors (excluding the worker itself) to 1 because it achieves extra consensus in $\log_2 n$ steps. There are also topology designs that can achieve good convergence rate with arbitrary n (not limited to powers of 2) [41] or achieve finite-time extra consensus with arbitrary n and arbitrary limitation on the number of neighbors [42].

Impractical assumption on communication cost While the theoretical analysis on the convergence performance of communication topologies in terms of the number of iterations has been well investigated, the fundamental assumption that the communication cost barely holds in practice. As introduced in Section 2.1, the

communication links are highly heterogeneous, and it is also the case when training with geo-distributed workers [43]. Therefore, whether the conclusion on the optimal convergence performance based on the number of iterations can not directly translate into optimal runtime efficiency.

2.3.4 Communication-computation patterns

Decentralized stochastic gradient descent (DSGD) interleaves local computation with neighbor communication to achieve both optimization and consensus. Depending on how communication is scheduled relative to local gradient computation, DSGD algorithms are commonly categorized into two communication-computation patterns: Adapt-Then-Combine (ATC) and Adapt-While-Combine (AWC). By denoting $X^{(t)} = [x_1^{(t)}, x_2^{(t)}, \dots, x_n^{(t)}]^\top$ and $G^{(t)} = [\nabla f_1(x_1^{(t)}), \dots, \nabla f_n(x_n^{(t)})]$, the matrix-form updates of AWC and ATC patterns can be written as

$$\begin{aligned} \text{Adapt-while-combine (AWC)} \quad X^{(t+1)} &= WX^{(t)} - \alpha G^{(t)} \\ \text{Adapt-then-combine (ATC)} \quad X^{(t+1)} &= W(X^{(t)} - \alpha G^{(t)}) \end{aligned} \quad (2.3.6)$$

Beyond the basic scheduling, several algorithmic variations have been proposed to address specific limitations of these two patterns:

Gradient tracking and bias correction Standard D-PSGD suffers from a steady-state error (bias) when data is heterogeneous (non-IID) across workers, preventing convergence to the exact global optimum unless the learning rate decays to zero. Algorithms like EXTRA [44] and DIGing [45] introduce a *gradient tracking* variable. Workers exchange both model parameters and a tracking vector that estimates the global average gradient. While these methods theoretically guarantee exact convergence in convex settings, the additional communication overhead (doubling the message size) often makes them less attractive for DNN training with i.i.d. data distributions, where stochastic noise usually dominates the sources of the consensus errors.

Local updates with/without outer optimizer Even though it is already challenging to achieve on-par training and evaluation performance as centralized methods in decentralized training with ATC or AWC patterns, efforts [46, 47] are made to further reduce the communication cost of decentralized training by local updates and fewer communication rounds. Another line of work [48, 49, 50] — similar to federated training or decentralized training with a fully connected graph — makes workers do several local updates, aggregates the difference from the last global synchronization, and then performs a global update using the difference as a surrogate gradient in the outer loop. They successfully achieved good runtime performance and facilitated fault tolerance with workers geographically widely distributed. The methods are interesting from the perspective that global computation resources can be utilized, but they might be too conservative if training on GPU clusters where the communication cost is not higher by orders of magnitude compared with computation.

2.3.5 Consensus errors in decentralized training

The prevailing perspective on decentralized training is that it should approximate synchronous/centralized training as closely as possible. To mitigate discrepancies among local models caused by weakly connected networks, prior work has focused on tracking global information [48, 51, 25, 52], enhancing communication topologies to improve convergence rates [40, 53, 41], and more. In addition, several theoretical studies [53, 54] establish a theoretical connection between the connectivity of decentralized communication topologies and both convergence and generalization, demonstrating that weaker connectivity results in poorer outcomes on both fronts.

2.3.6 Decentralized training framework

While theoretical research is abundant, production-grade frameworks for decentralized training are relatively under development compared to centralized solutions.

- BlueFog [27]: BlueFog is a comprehensive framework for decentralized training on HPC clusters, which is built upon the MPI standard and optimized with NCCL. BlueFog introduces neighbor-based collectives (e.g., `neighbor_allreduce`) into the PyTorch ecosystem. It supports dynamic topology management (e.g., switching between ring and exponential graphs) to accelerate consensus. The framework was under active development around 2020, but it is now out of update and not compatible with the latest version of PyTorch. As MPI and NCCL were integrated into `torch.distributed` as standard communication backends, the dependencies of BlueFog seem to over-complicate the setups.
- Bagua [28]: Bagua is a framework focusing on high-performance distributed training, and it also supports decentralized SGD. It rewrote the communication layer in Rust¹ to support both centralized communications with low-bit compression and AWC decentralized training scheme. The framework is efficient but at the cost of flexibility and complexity in APIs. Moreover, the framework focuses more on centralized training and lacks in showcase of decentralized training. Similar to BlueFog, the framework is not actively maintained since 2021.
- Flower [55], Hivemind [56]: Flower and Hivemind are two federated training frameworks which can be adapted for decentralized training. The focus of these frameworks is more on training with geographically distributed workers. The flexibility in internet communication comes at the cost of less optimized communication efficiency when applied on GPU clusters.

2.4 Sharpness and generalization in deep learning

The relationship between the geometry of the loss landscape — specifically “sharpness” or “flatness” of local minima — and the ability of a model to generalize to

¹<https://rust-lang.org/>

unseen data has been a central topic in deep learning theory. The prevailing hypothesis, often traced back to [57], is that **flat minima generalize better than sharp minima**. As shown in the conceptual plot in Figure 2.4.1, the intuition is that flat minima are more robust to shifts between the training and test error surfaces (distributional shift).

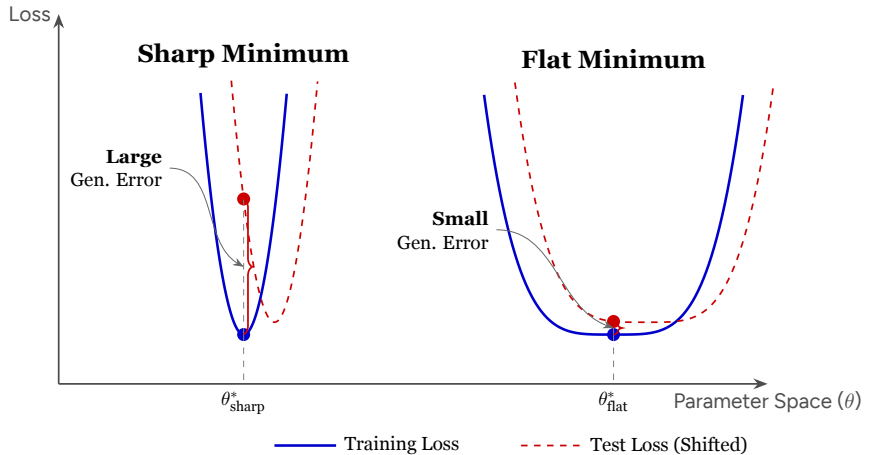


Figure 2.4.1: A conceptual plot of flat minimum and sharp minimum.

The “flatness” hypothesis The modern resurgence of this topic began with [34]. The work empirically demonstrated that large-batch stochastic gradient descent (SGD) tends to converge to sharp minima with poor generalization, while small-batch SGD finds flat minima with better generalization. This established a strong empirical link between the geometry of the solution and test performance.

The challenge: reparameterization invariance A critical counter-argument was raised by [58]. The work proved that for networks with ReLU activations, one can rescale weights (reparameterization) to make a minimum arbitrarily sharp or flat without changing the function’s output. This meant that standard sharpness metrics (like Hessian eigenvalues) could not be the sole cause of generalization, as the geometry could be manipulated independently of the model’s prediction.

Practical success: Sharpness-Aware Minimization Despite the theoretical issues raised by [58], the heuristic remained practically useful. Driven by the insights, the focus shifted from merely observing sharpness to actively minimizing it, beginning with the “worst-case” approach. [59] introduced Sharpness-Aware Minimization (SAM), an optimization algorithm that solves a min-max optimization problem (Table 2.4.2) with approximations. SAM seeks parameters that minimize the loss not at the current point, but at the worst point within a defined neighborhood. By explicitly penalizing the sharpest direction of ascent, SAM successfully forces the

model into flat basins, achieving state-of-the-art results across various domains. The work reinforced the practical utility of the flatness hypothesis.

Emerging variants of SAM To address the reparameterization issue, [60] proposed Adaptive Sharpness-Aware Minimization (ASAM). They introduced "adaptive sharpness", a measure that is invariant to scale, showing a stronger and consistent correlation with generalization than non-adaptive metrics.

Even though SAM and ASAM effectively improve the generalization performance, the main criticism lies in their computation cost. As shown in Table 2.4.2, both SAM and ASAM require one extra backpropagation to solve ϵ , which doubles its per-iteration computation cost compared with vanilla SGD. [61] proposed Low-Pass Filtering SGD (LPF-SGD) as a smoother, "average-case" alternative. Rather than fighting the single worst direction, LPF-SGD minimizes a Gaussian-smoothed version of the loss landscape. In practice, this is often approximated via Monte Carlo sampling (injecting noise), which allows it to filter out sharp, high-frequency minima without the explicit double-step of SAM. However, this efficiency comes with a trade-off. By acting "passively" (smoothing the landscape) rather than "actively" (counteracting the worst-case), LPF-SGD generally underperforms SAM in strict accuracy metrics. It improves over standard SGD, but rarely matches the strong regularization of min-max strategy in SAM.

There are more variants of SAM proposed recently [62, 63, 64, 65], which can either achieve slightly better generalization performance than SAM and ASAM or achieve better efficiency (between vanilla SGD and SAM) but with on-par or slightly worse generalization performance. Their ideas can be summarized as (1) use cheap approximations for ϵ , or (2) interpolate the sharpness loss with the original loss.

Those methods still approximately solve the objectives described in

Table 2.4.2.

	SAM [59]	ASAM [60]	LPF-SGD [61]
Objective	$\min_x \max_{\ \epsilon\ \leq \rho} F(x + \epsilon)$	$\min_x \max_{\ T_x^{-1}\epsilon\ \leq \rho} F(x + \epsilon)$	$\min_x \mathbb{E}_{\epsilon \sim \mathcal{N}(0, \sigma^2 I)} [F(x + \epsilon)]$
Definitions	$\epsilon^{(t)} = \rho \frac{\nabla F(x^{(t)})}{\ \nabla F(x^{(t)})\ _2}$	$\epsilon^{(t)} = \rho \frac{T_x^2 \nabla F(x^{(t)})}{\ T_x \nabla F(x^{(t)})\ _2}$ $T_x = \text{diag}(x_1^{(t)} , \dots, x_d^{(t)})$	$\epsilon^{(t)} \sim \mathcal{N}(0, \sigma^2 I)$
Update		$x^{(t+1)} = x^{(t)} - \alpha \nabla F(x^{(t)} + \epsilon^{(t)})$	

Table 2.4.2: Summary and comparison of SAM [59], Adaptive SAM [60] and LPF-SGD [61].

Recent skepticism and nuance In the last few years, the narrative has shifted again. Comprehensive studies by [66] and [67] have shown that sharpness metrics do not consistently correlate with generalization across all architectures and data regimes. They argue that while algorithms like SAM and LPF-SGD undeniably improve performance, they likely do so through a combination of mechanisms — such as inducing sparsity or specific feature biases — rather than solely by flattening the loss surface. The field has thus moved to a nuanced understanding: while seeking flat

minima (via worst-case or average-case methods) is a powerful heuristic, it is likely a proxy for deeper, more complex generalization dynamics.

2.4.1 Connection with decentralized SGD

An interesting perspective raised in [26] connects decentralized SGD and average-direction SAM (AD-SAM). If one treats the optimization process of decentralized SGD as the optimization on the (virtual) global average instead of a multi-agent optimization, each worker model can be viewed as a perturbed version of the global average center. Following the idea, the objective can be rewritten as

$$\begin{aligned}
 & J^{(t)}(x_1, \dots, x_n) \\
 &= \sum_{i=1}^n f_i(x_i^{(t)}) + \frac{1}{2\alpha^{(t)}} \sum_{i,j \in [n]} W_{ij} \|x_i^{(t)} - x_j^{(t)}\|^2 \\
 &= \underbrace{\sum_{i=1}^n f_i(\bar{x}^{(t)})}_{\text{objective on deployed model}} + \underbrace{\sum_{i=1}^n [f_i(x_i^{(t)}) - f_i(\bar{x}^{(t)})]}_{\text{sharpness}} + \underbrace{\frac{1}{2\alpha^{(t)}} \sum_{i,j \in [n]} W_{ij} \|x_i^{(t)} - x_j^{(t)}\|^2}_{\text{consensus regularizer}}
 \end{aligned} \tag{2.4.7}$$

With $\epsilon_i^{(t)} = x_i^{(t)} - \bar{x}^{(t)}$, the objective can be seen as (1) an objective on the deployed model, (2) a sharpness term similar to average-direction SAM with ϵ following the distribution defined by the weight diversity, and (3) the consensus regularizer.

In [26], the authors assume the consensus error is random perturbation and, with this assumption, prove the decentralized SGD is asymptotically equivalent to the average-direction SAM. However, the intrinsic structure of the consensus error is still under investigated, and the theoretical connection between DSGD and AD-SAM does not lead to substantial improvement in the generalization performance.

3 Preliminaries and Problem Setup

3.1 Notations

Throughout the thesis, we denote the set of integers from 1 to k by $[k] = \{1, \dots, k\}$. We let \mathbb{R}^d denote the vector space of d -dimensional column vectors with real entries. We denote the inner product of vectors and the norm of a vector by $\langle x, y \rangle = x^\top y$ and $\|x\|_2 = \sqrt{x^\top x}$, respectively. We then let $\mathbb{R}^{d \times n}$ denote the matrix space of $d \times n$ matrices, and the Frobenius norm of a matrix is denoted by $\|X\|_F = \sqrt{\text{tr}(X^\top X)}$.

In decentralized algorithms, we denote the number of workers by n , the local model of worker i at iteration t by $x_i^{(t)}$, and the local stochastic gradient of worker i at iteration t by $g_i^{(t)}$. We denote the mixing matrix by $W \in \mathbb{R}^{n \times n}$, and W is assumed to be non-negative, symmetric, and doubly stochastic. $W_{ij} > 0$ if worker i and worker j communicate with each other.

The consensus error $e_i^{(t)}$ is defined by the difference between the local model and the global average center, which is

$$e_i^{(t)} = x_i^{(t)} - \bar{x}^{(t)} = x_i^{(t)} - \frac{1}{n} \sum_j x_j^{(t)}.$$

3.2 Problem setup

The thesis focuses on investigating the potential of decentralized training in GPU clusters or data centers for improving the efficiency and generalization performance compared with typical data parallelism. Therefore, it is assumed that the data distributions among workers are i.i.d. Moreover, as the communication cost and computation cost are comparable under the setups in GPU clusters, and the homogeneous computation hardware is easily accessible, we focus on synchronous

decentralized training with one communication round per iteration. This scenario is also widely studied in many other literature [23, 40, 24, 1], and is important for improving the practicality of large-scale decentralized training.

3.2.1 Evaluation metrics

To assess the performance and efficiency of the decentralized training framework against baseline methods, we employ two primary categories of metrics: computational efficiency and generalization performance.

Efficiency metrics The primary metric for computational efficiency is the per-iteration runtime. To ensure a fair comparison across all methods, we maintain a constant number of total iterations for each experiment, excluding the overhead associated with dataset loading and pre-processing. Given a fixed iteration budget, the per-iteration runtime provides a direct and reliable estimation of the total training duration.

Generalization and model performance After training, models are evaluated on a held-out test set. The specific metrics used depend on the nature of the machine learning task:

- **Classification accuracy:** For image classification tasks, we report Top-1 and Top-5 accuracy. Top-1 accuracy measures the frequency with which the highest-probability prediction matches the ground truth, while Top-5 accuracy considers a prediction correct if the target label is within the five highest-probability categories.
- **Translation quality (BLEU and METEOR):** For neural machine translation (NMT) tasks, we utilize the BLEU [68] and METEOR [69] scores.

BLEU is a precision-oriented metric that calculates n-gram overlap between the generated and reference text, correlating strongly with human judgment at the corpus level.

METEOR provides a more nuanced evaluation by balancing precision and recall (with a higher weight on recall). It improves upon BLEU by incorporating semantic similarity, such as synonymy and stemming, leading to a better correlation with human judgment at the sentence level.

- **Cross-entropy loss:** We monitor the test loss for both classification and NMT tasks. While not a direct measure of task-specific performance, it provides insight into how well the model optimizes the objective function on unseen data.
- **Training loss and generalization gap:** We record the training loss to monitor how well the model fits the training distribution. By analyzing the generalization gap — the difference between test loss and training loss — we can assess the degree of overfitting and the robustness of the decentralized training process.

Practical remarks on distributed data sampler The common practice for the distributed data sampler (also used in our experiments) is to reshuffle the full dataset at the start of each epoch and partition it evenly across workers. The strategy ensures, in expectation, i.i.d. data distributions among workers.

4 Efficient decentralized learning

4.1 Overview

We explore the potential of decentralized training as a practical alternative to conventional All-Reduce-based data-parallel training, particularly in communication-constrained environments. Instead of aggregating global gradient at every iteration to ensure the consistency, we allow local models to evolve semi-independently and communicate through decentralized communication topologies. This design reduces synchronization overhead and improves hardware utilization.

This chapter is built on our work presented in *From Promise to Practice: Realizing High-Performance Decentralized Training* [1]. Decentralization introduces significant complexity to the design space of the distributed training, and the theoretical benefits of decentralized training may not directly translate into speedup with on-par generalization performance in practical experiments. We investigate when and why decentralization yields efficiency gains, develop a runtime model that predicts these gains, and demonstrate the superior runtime performance of decentralized training under various hardware setups. Through this chapter, we bridge the gap between theoretical potential and real-world efficiency, providing a systematic framework for understanding the runtime performance of decentralized training.

4.2 Key factors for efficient multi-node decentralized training

We identify three key factors that determine whether decentralized learning is practically efficient: (1) overlapping communication with computation, (2) accounting for heterogeneous communication costs, and (3) mitigating sensitivity to computation variance across workers. Together, these elements govern how decentralized methods can outperform centralized ones in total runtime.

4.2.1 Enhance overlap between communication and computation

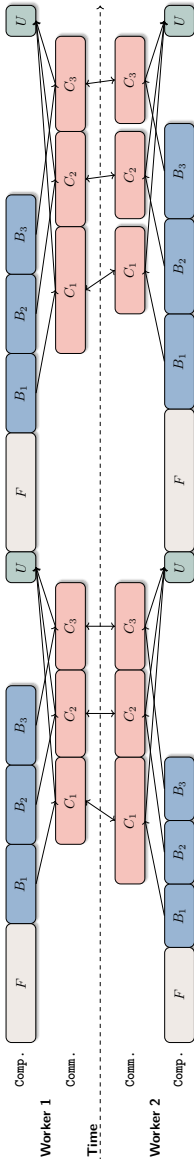


Figure 4.2.1: Example timeline of All-Reduce training. F : forward pass, B_i : backward pass for bucket i , C_i : gradient communication for bucket i , and U : optimizer update. The arrows represent the dependencies.

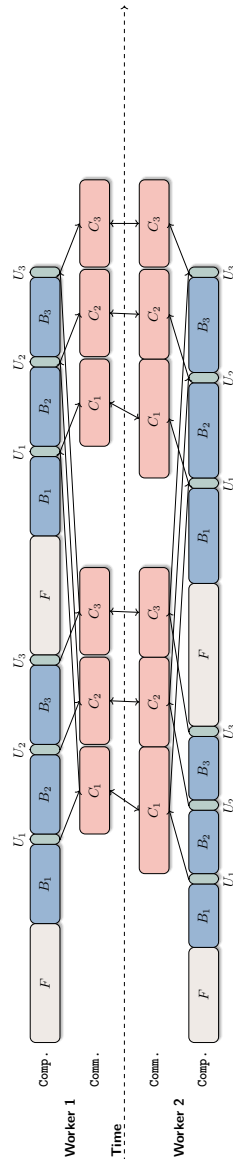


Figure 4.2.2: Example timeline of the adapt-while-communicate decentralized training. F : forward pass, B_i : backward pass for bucket i , C_i : parameter communication for bucket i , and U_i : optimizer update for bucket i . The arrows represent the dependencies.

Modern accelerators allow concurrent communication and computation, but traditional All-Reduce synchronization limits this overlap. In All-Reduce training, each worker must wait for a complete aggregation of gradients before proceeding to the next iteration, particularly for the last bucket in backpropagation. The example in Figure 4.2.1 demonstrates that, even with the gradient bucketing technique, the communication time of the last bucket can not be hidden by the computation, which causes idling in the computation GPU stream.

Decentralized training, by contrast, further enables overlap even with the forward pass through model averaging and asynchronous updates. The general update rule of the adapt-while-communicate decentralized training is

$$x_i^{(t+1)} \leftarrow -\alpha^{(t)} d(x_i^{(t)}) + \sum_{j \in \mathcal{N}_i^{(t)}} w_{ij}^{(t)} x_j^{(t)},$$

where $d(x_i^{(t)})$ denotes the update that depends on the local model only. In this scheme, each worker updates its parameters based on its local model and neighbor information from the last iteration, and it is independent of neighbor information from the current iteration. By applying the idea of the gradient bucketing to model parameters, this independence allows asynchronous execution: after one bucket is updated, communication for the current bucket can be launched immediately, and the communication result is not required until the next iteration. The pattern leads to reduced idle time and higher hardware utilization compared to All-Reduce training. In the example in Figure 4.2.2, workers in decentralized training can achieve 100% utilization on the computation thread. Compared with All-Reduce training (Figure 4.2.1), decentralized training achieves a significant speedup in this 2-iteration example.

4.2.2 Respect heterogeneous communication costs

Another key advantage of well-designed decentralized training over All-Reduce training is the reduced communication cost due to the sparsity of decentralized communication. Unlike All-Reduce training, where all workers must participate the global aggregation communication in each iteration, decentralized training allows workers to communicate only with selected neighbors. While the advantage is appealing, the decentralized averaging causes discrepancies among local models. To close the gap between decentralized methods and centralized methods, researchers have invested great efforts in designing the decentralized communication topology [70, 71, 72, 40, 73]. However, these works simply assume the communication costs are uniform in the network and consider the convergence rates of the topologies in terms of the number of iterations.

As introduced in Section 2.1, the assumption does not reflect the actual hardware architecture, and the communication bandwidth and latency differ markedly between intra-node and inter-node connections. To demonstrate the practical impact of topology and heterogeneous connections on communication time, we set up an experiment with two scenarios: four nodes, each equipped with four GPUs, are interconnected by either 25Gbps Ethernet or 100Gbps Infiniband. Table 4.2.1 shows the communication time of performing averaging operations with different

Setups \ Topology	Complete	One-peer Exp.	One-peer Ring	Alternating Exp Ring
$4 \times 4 \times A100, 100\text{Gbps}$	55.31 s	55.44 s	32.97 s	29.55 s
$4 \times 4 \times A40, 25\text{Gbps}$	395.99 s	429.47 s	217.90 s	185.93 s

Table 4.2.1: Communication time of performing averaging operations for 8192 iterations over three 25MB FP32 tensors. **Setup 1:** 16 A100 GPUs on 4 nodes inter-connected by 100Gbps Infiniband. **Setup 2:** 16 A40 GPUs on 4 nodes inter-connected by 25Gbps Ethernet. Here, Complete means global averaging.

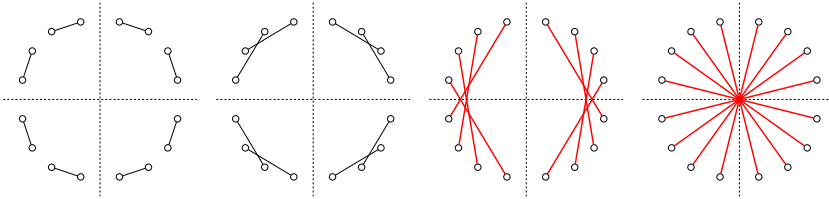


Figure 4.2.3: The communication operations in each iteration for the one-peer exponential topology. The circles are the GPUs, the dashed lines are the boundaries of the nodes, the lines connecting the circles are the communication operations, and the lines in red are the communication operations that span across multiple nodes.

communication topologies. Even though the One-peer Exponential topology enjoys rapid convergence of consensus errors in terms of iterations, it requires more time than the Complete topology in this setup due to frequent inter-node communication. The results are expected by viewing the actual communication operations in every iteration. Figure 4.2.3 demonstrates that, even though there are two iterations with purely intra-node communication, there are two iterations where all the communication operations are inter-node, which causes congestion at the switch and increases the average communication time.

To adapt to the communication imbalance in the two setups used in our experiments, we design the Alternating Exponential Ring (AER) topology. The efficiency advantages in AER topology are: (1) there is only one inter-node communication at maximal for each node, which avoids the bottlenecks limited by the NIC of the node, (2) there is only one inter-node communication among all the nodes, which avoids the congestion at the switch, (3) the workers in a same communication group are implicitly synchronized by the balanced communication costs, and (4) the intra-node communication links are fully utilized. As shown in Figure 4.2.4, AER topology only has one communication operation that spans on two nodes in each iteration, and each worker only Table 4.2.1 shows AER topology achieves superior runtime performance. Moreover, as demonstrated in Figure 4.2.5, AER topology also achieves a better convergence rate than the one-peer exponential topology.

Thus, when designing a communication topology for decentralized training in such environments, it is crucial to consider both the connectivity, which affects the speed of information propagation, and the heterogeneity of networking connections, which impacts the execution time of each communication round. It is also important to note

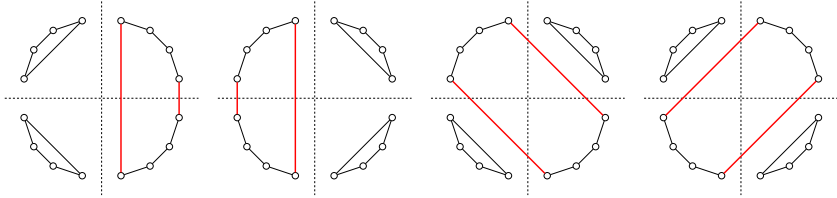


Figure 4.2.4: The communication operations in each iteration for the alternating exponential ring topology. The circles are the GPUs, the dashed lines are the boundaries of the nodes, the lines connecting the circles are the communication operations, and the lines in red are the communication operations that span across multiple nodes.

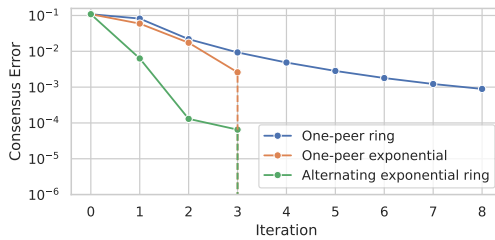


Figure 4.2.5: Convergence comparison of various topologies with 16 workers. The initial values of the workers are randomly sampled from a normal distribution.

that the effectiveness of a specific topology may vary depending on the operational environment. By quantifying the communication costs associated with different connections and the congestion caused by concurrent communications, the formulation of an optimization problem that targets both the convergence rate and average runtime as objectives is a direct future work. However, in the world of deep learning which is highly non-convex, whether a better convergence rate of the topology is needed remains underexplored (see Section 5.3).

4.2.3 Reduce sensitivity to stragglers

Even in homogeneous hardware environments, variations in computation time can occur due to workload imbalance or system noise [74]. In Figure 4.2.6, we demonstrate the distributions of computation time in two classical tasks. In the neural machine translation task, where the workloads may vary across iterations, the variance in the computation time is significant. Even in the image classification task, where the workloads are perfectly balanced in all iterations, the variance is still observable.

All-Reduce training is particularly sensitive to such stragglers because the gradient aggregation in each iteration must wait for the slowest worker. If the distributions of the computation time of workers are i.i.d. and are the normal distribution, the expected value of the maximum of the N computation times is asymptotically $\Theta(\sqrt{\log N})$ [33]. In the example in Figure 4.2.1, worker 2 has a shorter computation

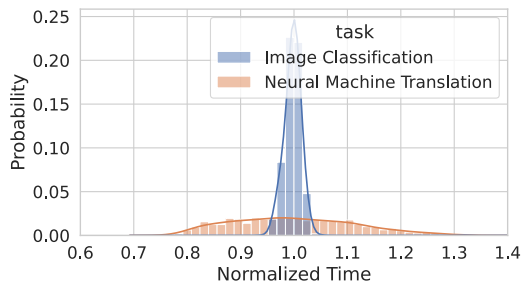


Figure 4.2.6: Distributions of the normalized computation time in two tasks. Image classification: ResNet50 on ImageNet. Neural machine translation: Transformer (base) on WMT14 English-German.

time for the first iteration, then it has to take additional idling time to synchronize with worker 1.

In decentralized training, however, only neighboring workers synchronize, and the update is independent of neighbor information in the current iteration, allowing computational resources to proceed independently. In the example in Figure 4.2.2, even though worker 2 is faster in the first iteration, the communication time is still fully hidden because of the asynchronous update.

It should also be noted that, in other implementations of decentralized training like SGP [23]¹, they group the intra-node workers as one worker in the decentralized training. For the intra-node workers, they perform gradient aggregation within the node then perform decentralized communication across the nodes. Even though intra-node gradient aggregation can be fast due to low-latency, high-bandwidth connections, it can still suffer from the stragglers as the case in All-Reduce training. Our proposed design further enhances resilience by decoupling gradient synchronization within nodes, avoiding intra-node bottlenecks.

4.3 Runtime model for predicting speedup

In the last section, three key factors enabling the superior speedup of decentralized training are analyzed. However, when the decentralized training can achieve speedup and how much the speedup can be are still under-investigated.

A central contribution of this study is a simple yet predictive runtime model that quantifies the key factors and gives a decent prediction on the per-iteration speedup.

For basic setups, we have n workers, and the model is divided evenly into b buckets. We assume the global batch size is fixed. Then we set the time taken by the forward pass on one bucket with one worker as 1 unit time, and the time taken by a backward pass on one bucket with one worker as 2 unit time (a reasonable approximation as

¹https://github.com/facebookresearch/stochastic_gradient_push

described in [75] and from our profiling results). We also denote the time that it takes to update one bucket by θ unit time where θ is the same for All-Reduce training and decentralized training and independent of the number of workers. When $\sigma^2 = 0$, we assume the workload is evenly distributed among workers and every worker process $1/n$ of the global batch, which means the computation cost scales as $1/n$.

The model quantifies the following key factors:

- $\gamma \in (0, \infty)$: is the time of a global All-Reduce communication of one bucket. Note that γ is the time spent on the communication, while the actual time of the All-Reduce operation could be longer due to synchronization across workers. Moreover, with the computation time of the forward pass is set to unit time, γ is actually a relative indicator.

The γ parameter captures the combination of several system characteristics:

1. Network technology. A worse network connection leads to a slower global All-Reduce and a larger γ .
 2. Number of workers. A larger number of workers potentially lead to a slower All-Reduce and a higher value of γ .
 3. Arithmetic intensity. Arithmetic intensity is the ratio of the number of arithmetic operations to the memory footprint. A smaller global batch size potentially decreases the arithmetic intensity. It is also relevant to model architectures. For example, convolution neural networks have higher arithmetic intensity than transformer models in general. A lower arithmetic intensity leads to a higher value of γ .
- $\omega \in (0, 1]$: the ratio of the time of a decentralized communication round and an All-Reduce operation. Note that ω can exceed one, but in that case it is always better to use All-Reduce (as discussed in Section 4.2.2). Using a more sparse communication topology could potentially decrease ω .
 - $\sigma^2 \in (0, \infty)$: the variance in the truncated normal distributions used to model the variance (imbalanced workload) in the computation time. For worker i at iteration t , a random number $p_i^{(t)}$ is sampled from the truncated normal distribution with mean 1, variance σ^2 , and support $[0.5, 1.5]$ (chosen to match our experimental results in Figure 4.2.6). The random number is used as a multiplier for the computation time of worker i at iteration t , *i.e.* the time for both forward and backward passes are multiplied by the same $p^{(i,t)}$.

With these quantities, we further take the dependency relationship into consideration. We denote the completion time of task X at worker i in iteration t by $T_X^{(i,t)}$. Then,

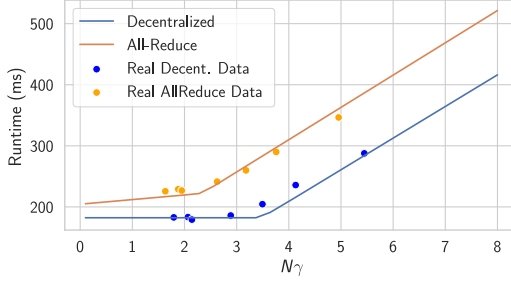


Figure 4.3.7: Runtime model and actual runtime in the experiment of training Transformer (base) on WMT14 English-to-German.

based on the dependency relations in All-Reduce training, we have

$$\begin{aligned}
 T_F^{(i,t)} &= T_U^{(i,t-1)} + p^{(i,t)} \frac{b}{n} \\
 T_{B_k}^{(i,t)} &= \begin{cases} p^{(i,t)} \frac{2}{n} + T_F^{(i,t)} & \text{if } k = b \\ p^{(i,t)} \frac{2}{n} + T_{B_{k+1}}^{(i,t)} & \text{if } k \in \{1, \dots, b-1\} \end{cases} \\
 T_{C_k}^{(i,t)} &= \begin{cases} \gamma + \max_{j \in [n]} \{T_{B_k}^{(j,t)}\} & \text{if } k = b \\ \gamma + \max_{j \in [n]} \{T_{B_k}^{(j,t)}, T_{C_{k+1}}^{(j,t)}\} & \text{if } k \in \{1, \dots, b-1\} \end{cases} \\
 T_U^{(i,t)} &= T_{C_1}^{(i,t)} + \theta b
 \end{aligned} \tag{4.3.1}$$

where all out-of-bound finish times like $T_U^{(i,0)}$ are defined to be zero.

To state similar dependency relations in decentralized training, we denote the set of neighbors of worker i in iteration t as $\mathcal{N}_i^{(t)}$. Then, we have

$$\begin{aligned}
 T_F^{(i,t)} &= T_{U_1}^{(i,t-1)} + p^{(i,t)} \frac{b}{n} \\
 T_{B_k}^{(i,t)} &= \begin{cases} p^{(i,t)} \frac{2}{n} + T_F^{(i,t)} & \text{if } k = b \\ p^{(i,t)} \frac{2}{n} + T_{U_{k+1}}^{(i,t)} & \text{if } k \in \{1, \dots, b-1\} \end{cases} \\
 T_{U_k}^{(i,t)} &= \max\{T_{B_k}^{(i,t)}, T_{C_k}^{(i,t-1)}\} + \theta \\
 T_{C_k}^{(i,t)} &= \begin{cases} \omega\gamma + \max_{j \in \mathcal{N}_i^{(t)}} \{T_{U_k}^{(j,t)}, T_{C_1}^{(j,t-1)}\} & \text{if } k = b \\ \omega\gamma + \max_{j \in \mathcal{N}_i^{(t)}} \{T_{U_k}^{(j,t)}, T_{C_{k+1}}^{(j,t)}\} & \text{if } k \in \{1, \dots, b-1\} \end{cases}
 \end{aligned} \tag{4.3.2}$$

where all out-of-bound finish times like $T_{U_1}^{(i,0)}$ and $T_{C_k}^{(i,0)}$ are defined as zeros.

Using these parameters, the model analytically characterizes how per-iteration runtime grows with communication delay and computation variance. As shown in Figure 4.3.7, by measuring the quantities based on the profiling results, the runtime model accurately captures the trend of the actual runtime, which implies its

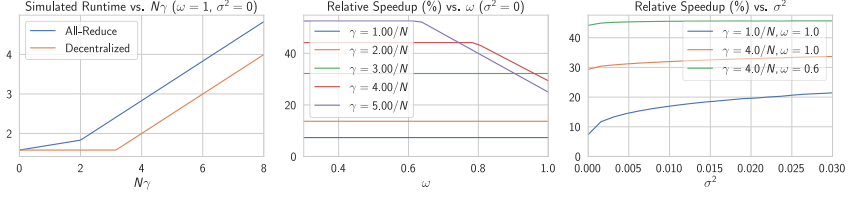


Figure 4.3.8: Simulation by the runtime model. In the simulation, $N = 8$ (8 workers), $b = 4$ (4 buckets, from ResNet-50 experiments), $\theta = 0.05$ (time taken by updating one bucket is 0.05 unit time), and the communication topology of decentralized training used for determining the neighbors is Complete topology.

practicality in improving the understanding of the speedup brought by decentralized training.

With the runtime model, we can derive useful interpretations and guidelines for an efficient implementation of decentralized training:

Closed-form per-iteration runtime estimation when $\sigma^2 = 0$ When the computation variance is negligible ($\sigma^2 = 0$), the computation time is deterministic, then we can derive the per-iteration runtime for All-Reduce training and decentralized training as follows:

$$T_{\text{All-Reduce}} = \begin{cases} \frac{3b}{n} + \theta b + \gamma, & \gamma \in (0, \frac{2}{n}], \\ \frac{b+2}{n} + \theta b + b\gamma, & \gamma > \frac{2}{n}. \end{cases} \quad (4.3.3)$$

where the critical point $\gamma = 2/n$ indicates the case where the time of the global All-Reduce can be barely hidden by the backward computation. With $\gamma \in (0, 2/n]$, the runtime increases with γ because the communication time of the last bucket cannot be hidden. After $\gamma > 2/n$, the per-iteration runtime is dominated by the global All-Reduce communication.

$$T_{\text{Decentralized}} = \begin{cases} \frac{3b}{n} + \theta b, & \omega\gamma \in (0, \frac{3}{n} + \theta], \\ b\omega\gamma, & \omega\gamma > \frac{3}{n} + \theta. \end{cases} \quad (4.3.4)$$

where the critical point $\omega\gamma = \theta + 3/n$ indicates the point where the time of the decentralized communication can be barely hidden by the whole computation pass (forward, backward and update). Compared with All-Reduce training (Eq. 4.3.3), the per-iteration runtime of decentralized training keeps constant with $\omega\gamma \in (0, \theta + 3/n)$ because the computation resources are fully utilized before the communication time can not be hidden by the whole computation pass. Similar to All-Reduce training, the runtime is dominated by the communication after the critical point, while decentralized training still can outperform All-Reduce training by some constants because decentralized training further allows the overlap between the forward pass and the communication.

The left panel in Figure 4.3.8 demonstrates the trends and the critical points captured by the closed-form estimation. In short, decentralized training can always achieve superior runtime performance than All-Reduce training even with $\omega = 1$.

Best-possible speedup achieved by decentralized training With the closed-form runtime estimation, we can derive an important closed-form expression for the best possible speedup $S(\gamma)$ when computation variance is negligible ($\sigma^2 = 0$) is:

$$S(\gamma) = \begin{cases} 1 + \frac{n\gamma}{b(3+n\theta)}, & \gamma \in (0, 2/n], \\ 1 + \frac{b(n\gamma-2)+2}{b(3+n\theta)}, & \gamma > 2/n. \end{cases} \quad \text{with } \omega \leq (3+n\theta)/(n\gamma) \quad (4.3.5)$$

In the closed-form expression for the best possible speedup, the condition $\omega \leq (3+n\theta)/(n\gamma)$ indicates the case where the communication time is fully hidden by computation in decentralized training. Since $T_{\text{Decentralized}}$ will not decrease further with smaller ω , decentralized training achieves best speedup in this case.

From the expressions, we can conclude that

- Decentralized training can achieve better $S(\gamma)$ with larger γ or, equivalently, slower communication speed (relative to computation) because $S(\gamma)$ is monotonically increasing with γ .
- Decreasing ω reduces the time for a bucket communication in the decentralized setting, which effectively moves us to the left along the decentralized runtime graph (Figure 4.3.8, left). This leads to increasing speed-ups until $\omega = (3+n\theta)/(n\gamma)$, after which no runtime improvements are obtained by decreasing ω further.

The middle plot in Figure 4.3.8 clearly demonstrates and supports our conclusions.

Decentralized training is more resilient to stragglers With $\sigma^2 > 0$, it is difficult to derive a close-form expression for the runtime for decentralized training. In the right plot in Figure 4.3.8, we simulate the model and perform Monte Carlo simulations to predict the speedup brought by decentralized training.

The three combinations of γ and ω represent the scenarios that (a) computation-bound ($\gamma = 1/n, \omega = 1$) (b) communication-bound ($\gamma = 4/n, \omega = 1$) (c) communication-bound and using a decentralized communication topology ($\gamma = 4/n, \omega = 0.6$). In the three scenarios, increasing σ^2 brings moderate improvement in the speedup of decentralized training.

Limitation of the runtime model While the runtime model has included many factors reflecting the real system, it is still not a perfect predictor for the runtime efficiency. For example, the model assumes perfectly uniform buckets, the model does not consider the variance in communication time, and the relative cost between the forward pass and the backward pass may vary across models and hyperparameter setups. Even though the model is not perfect, it correctly captures the dominant factors, which gives useful insights about ordering and turning points.

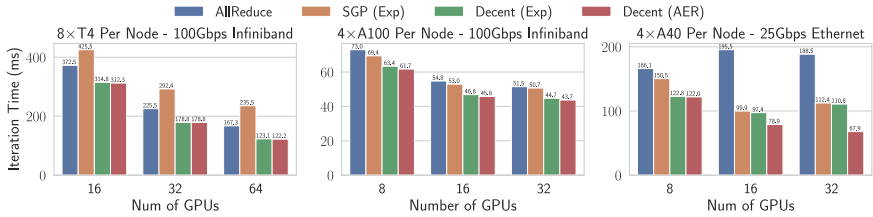


Figure 4.4.9: The results show the strong scaling performance (fixed global batch size, 25K tokens) of All-Reduce, 1-OSGP [23] and our implementations by reporting the per-iteration runtime with various numbers of workers and network settings. Here, Exp and AER stand for one-peer exponential graph and alternating exponential ring, respectively.

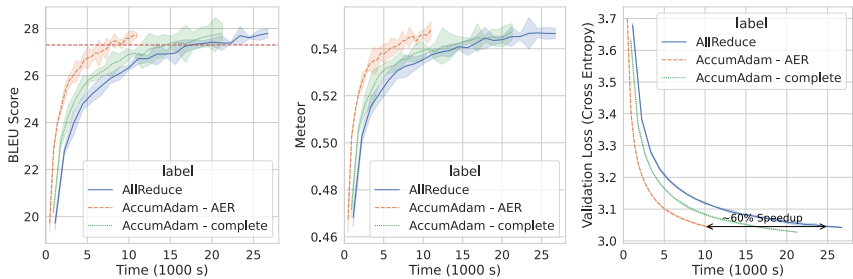


Figure 4.4.10: Training a transformer (base) model on the English-to-German translation task on four 4xA40 nodes with 25Gbps Ethernet connections. The error bands of $\pm 2\sigma$ are based on three runs. The red dashed line on the left is the baseline (27.3 BLEU score) from [31].

4.4 Numerical experiments

In this section, we compare decentralized training with All-Reduce training and assess the practical training-time speedups and generalization performance in cluster environments. We consider three large-scale tasks: neural machine translation, image classification, and GPT-2 pre-training. In all the experiments, we keep the number of iterations equal for both All-Reduce training and decentralized training. The baseline, All-Reduce training, uses Adam-based optimizers for all experiments. Detailed descriptions of all tasks, experimental setups, and hyperparameter settings can be found in [1, Appendix A.6].

Note that, in the experiments, the best generalization performance is achieved by *AccumAdam* which is an engineering stabilization mechanism and a variant of decentralized Adam proposed by us. With *AccumAdam*, the decentralized training achieves comparable generalization performance as All-Reduce training within the same iteration budget. We defer its introduction to the next chapter, which focuses on the generalization performance of decentralized training.

Node Setup / Inter-Connection	Workers	Methods	Per-iteration Runtime (ms)	Speedup
4×A40 Node / 25Gbps Ether.	8	All-Reduce	344.44	-
		Decentralized	333.07	3.30%
	16	All-Reduce	176.72	-
		Decentralized	163.68	7.38%
8×T4 Node / 100Gbps Infini.	16	All-Reduce	645.59	-
		Decentralized	617.75	4.31%
	32	All-Reduce	331.51	-
		Decentralized	313.56	5.41%

Table 4.4.2: Results of speedup of training ResNet-50 on ImageNet dataset for the image classification task. The topology is complete graph for decentralized training.

4.4.1 Neural machine translation on WMT14 with Transformer

For neural machine translation, following [31], we trained the transformer ($\sim 65\text{M}$ parameters for base variant and $\sim 213\text{M}$ parameters for big variant) on the WMT14 English-German and English-French dataset [76]. In each run, we evaluate the trained model by BLEU [68] and METEOR [69] on the test set.

Figure 4.4.9 demonstrates the superior scaling performance under various setups compared with baselines, All-Reduce training and SGP [23]. Figure 4.4.10 shows a $\sim 60\%$ speedup brought by decentralized training by presenting the curves of evaluation metrics with respect to the training time with 4×4 A40 GPUs.

4.4.2 Image classification on ImageNet with ResNet

In the image classification task, we trained ResNet-50 [77] on ImageNet-1K [78]. In each run, we evaluate the trained model by Top-1 and Top-5 accuracies on the validation set.

Table 4.4.2 shows the speedup brought by decentralized training. Since the computation time variance is small and the communication is relatively fast (convolutional neural networks have high arithmetic intensity), using sparser communication topologies does not bring more speedup. We report the results with the complete topology, and decentralized training still demonstrate non-trivial speedup compared with All-Reduce training.

4.4.3 LLM pretraining on OpenWebText with GPT-2

For the GPT-2 pretraining task, we trained GPT-2 (small) [79] on OpenWebText [80], and we report the training and validation losses of the trained models.

As shown in Figure 4.4.11, compared to All-Reduce training, decentralized training with AccumAdamW and the Alternating Exponential Ring topology achieves

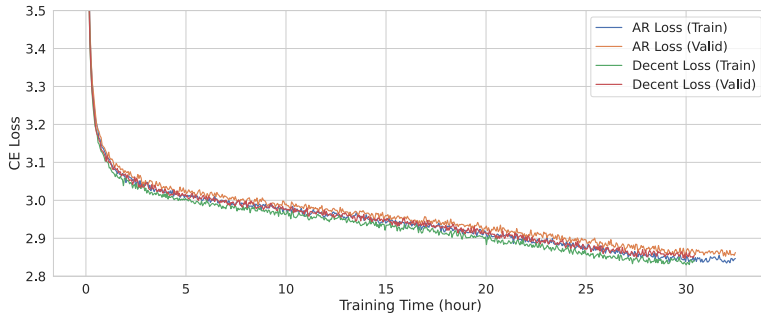


Figure 4.4.11: The curves of train and validation losses where the x-axis is training time. AR: All-Reduce training. Decent: Decentralized training with AccumAdamW.

comparable training loss (2.8419 vs. 2.8468), comparable validation loss (2.8561 vs. 2.8626), and superior runtime (30.41 hours vs. 32.461 hours) after 600k training steps on OpenWebText [80] using $4 \times 4 \times A100$ GPUs. Even though the workloads are balanced among the workers, it still achieves a relative speedup of 6.31%. More speedup can be expected by replacing the sequence packing with other batching strategies that may improve the performance but introduce imbalance in workloads [37, 36].

4.5 Summary

This chapter investigates the runtime efficiency of decentralized learning as a practical alternative to All-Reduce-based distributed training. We identify and analyze three determinants of efficiency, (1) communication–computation overlap, (2) heterogeneous communication costs, and (3) robustness to computation variance, and demonstrate how decentralized methods exploit these properties to achieve higher hardware utilization. A predictive runtime model is developed to characterize performance across varying network conditions and compute variability, accurately matching empirical results. Experiments on large-scale neural machine translation, image classification, and language modeling tasks confirm that decentralized training attains comparable accuracy to All-Reduce while reducing training time by 5–60%, depending on network conditions. The results provide a quantitative framework and design guidelines for efficient and robust decentralized learning systems.

Next While efficiency determines the practical viability of decentralized learning, it represents only one facet of successful large-scale training. Achieving robustness, ensuring that decentralized methods match or surpass the generalization performance of centrally synchronized training, is equally essential. The next chapter addresses this complementary aspect by examining the optimization stability and generalization behavior of decentralized training and by introducing algorithmic designs, such as *AccumAdam*, that enable robust convergence and competitive model quality.

5 Robust Decentralized Training

5.1 Overview

While the previous chapter focused on *efficiency* — identifying when decentralized training can be faster — this chapter turns to a subtler and arguably more fundamental question: how does decentralization affect generalization?

Early works on decentralized optimization were primarily motivated by system-level concerns: reducing communication bottlenecks, improving scalability, and avoiding synchronization barriers. Rooted in the convex problems, the canonical underlying assumption was that the closer a decentralized algorithm behaves to its centralized counterpart, the better. In this traditional view, the goal of algorithmic design was to approximate the global information locally and minimize consensus errors so that the decentralized model trajectory would nearly coincide with the trajectory of its centralized counterpart. Under this principle, we developed AccumAdam, a decentralized variant of Adam that explicitly accumulates local gradients to approximate centralized momentum statistics.

By eliminating the consensus errors, decentralized training is proven to achieve slightly worse or comparable generalization performance in both training and testing. However, this long-standing idea also limits the potential of decentralized training under centralized training, which makes decentralized training an efficient but defective alternative in terms of generalization.

As decentralized deep learning matured, a striking pattern began to emerge: perfect consensus does not always yield the best generalization. Recent investigations revealed two intriguing observations:

- there exists a threshold of consensus error below which generalization is unaffected [24], and

- decentralized SGD can actually outperform centralized SGD in generalization but only in large-batch settings [26].

These results suggest that consensus errors — long considered undesirable noise—may act as a form of implicit regularization. In other words, moderate disagreement among workers might bias the optimization trajectory toward minima that generalizes better.

This realization motivated a conceptual shift in our research: rather than eliminating consensus errors, can we control and exploit them? To investigate this question, we proposed DSGD-AC, a decentralized SGD algorithm that maintains bounded but non-diminishing consensus errors during training. The results show that, under certain conditions, moderate consensus discrepancies can drive the system toward flatter minima and yield better generalization.

This chapter traces this transition in understanding — from emulating centralization (AccumAdam) to embracing structured decentralization (DSGD-AC). We first revisit the design of AccumAdam, which closes the optimization gap between decentralized and centralized training (Section 5.2). We then introduce DSGD-AC and demonstrate how controlled consensus diversity enhances robustness (Section 5.3). Finally, we present empirical and theoretical evidence that consensus errors are not random noise but directional perturbations correlated with the curvature of the loss landscape (Section 5.3.4).

5.2 Alleviate the gap caused by decentralization

5.2.1 Decentralized Adam

To facilitate the training of transformer-based and other large-scale models, we introduce a decentralized Adam optimizer (DAdam) that preserves the key benefits of the classical Adam algorithm [81], adaptive learning rates and momentum, while supporting parallel computation-communication overlap.

Algorithm 1 Decentralized Adam on worker i

```

1:  $m_i^{(0)}, v_i^{(0)} \leftarrow 0; x_i^{(0)} \leftarrow x^{(0)}$ 
2: for  $t = 1, 2, \dots, T$  do
3:    $g_i^{(t)} \leftarrow \nabla \ell(x_i^{(t-1)}; \xi_i^{(t)})$ 
4:    $m_i^{(t)} \leftarrow \beta_1 m_i^{(t-1)} + (1 - \beta_1) g_i^{(t)}$ 
5:    $v_i^{(t)} \leftarrow \beta_2 v_i^{(t-1)} + (1 - \beta_2) [g_i^{(t)}]^2$ 
6:    $x_i^{(t)} \leftarrow -\alpha \frac{m_i^{(t)} / (1 - \beta_1^t)}{\sqrt{v_i^{(t)} / (1 - \beta_2^t) + \epsilon}} + \sum_{j \in \mathcal{N}_i} w_{ij} x_j^{(t-1)}$ 
7: end for

```

In Algorithm 1, each worker computes local gradients, updates its momentum statistics independently, and exchanges model parameters with neighbors according to the mixing matrix $W = [w_{ij}]$. Crucially, the local momentum variables m_i and v_i

depend only on the worker's local gradients, not on its neighbors' models. This separation enables communication–computation overlap, significantly improving efficiency in distributed GPU clusters (see Section 4.2.1).

5.2.2 Convergence analysis of decentralized Adam

We now establish the convergence of Algorithm 1 under standard assumptions.

Assumption 1 (*L-Lipschitz continuous function*) For each worker i , the gradient ∇F_i is L -Lipschitz continuous:

$$\|\nabla F_i(x) - \nabla F_i(y)\|_2 \leq L\|x - y\|_2, \quad \forall x, y \in \mathbb{R}^d.$$

Assumption 2 (*Globally bounded gradient*) There exists a constant $R \geq \sqrt{\epsilon}$ such that $\|\nabla \ell(x; \xi_i)\|_\infty \leq R - \sqrt{\epsilon}$ almost surely.

Assumption 3 (*Mixing matrix*) The mixing matrix $W = [w_{ij}]$ is non-negative and doubly stochastic. We denote $\lambda = \max_{i \in \{2, \dots, n\}} \{|\lambda_i|\} \in [0, 1)$. For simplicity, we further assume W is symmetric.

Assumption 4 (*i.i.d. data distributions*) All workers sample from the same data distribution, which means $F_1 = F_2 = \dots = F_n$.

Assumptions 1 and 2 are used in [82] to provide a simple analysis for centralized Adam. Assumption 3 is standard in decentralized learning and holds when the underlying communication topology is connected. Since we focus on decentralized training on GPU clusters, we further assume Assumption 4. Under these assumptions, we have the following convergence guarantee.

Theorem 1 Let τ be a random index drawn from $0, \dots, T - 1$ with

$$\mathbb{P}[\tau = \tilde{t}] \propto 1 - \beta_1^{T-\tilde{t}}$$

and define $\bar{x}^{(t)} = \frac{1}{N} \sum_i x_i^{(t)}$. Then, under $0 < \beta_1 < \beta_2 < 1$, Algorithm 1 satisfies:

$$\mathbb{E} \left[\|\nabla F(\bar{x}^{(\tau)})\|_2^2 \right] \leq \frac{4R}{\alpha \tilde{T}} (F(\bar{x}^{(0)}) - F_*) + E \left[\frac{1}{\tilde{T}} \ln \left(1 + \frac{R}{\epsilon(1 - \beta_2)} \right) - \frac{T}{\tilde{T}} \ln(\beta_2) \right],$$

where

$$\begin{aligned} E := & \frac{24dR^2\sqrt{1 - \beta_1}}{\sqrt{1 - \beta_2}(1 - \beta_1/\beta_2)^{3/2}} + \frac{2\alpha dLR(1 - \beta_1)}{(1 - \beta_2)(1 - \beta_1/\beta_2)} \\ & + \frac{4\alpha^2 dL^2\beta_1}{(1 - \beta_1/\beta_2)(1 - \beta_2)^{3/2}} + \frac{8\alpha^2(1 + \lambda^2)RL^2d\sqrt{1 - \beta_1}}{(1 - \lambda^2)^2(1 - \beta_2)(1 - \beta_1/\beta_2)\sqrt{\epsilon}} \end{aligned}$$

We refer readers to [1, Appendix B.4] for the proof.

Discussion Compared to the single-machine Adam analysis in [82], the error bound is nearly identical—differing only by a constant factor of 2 and an additional term proportional to α^2 (last term in E). For diminishing step sizes $\alpha = \alpha_0/\sqrt{T}$, this extra term vanishes. Thus, Decentralized Adam achieves the same asymptotic convergence rate as centralized Adam, confirming that consensus coupling does not fundamentally impair convergence.

5.2.3 Improved variant: AccumAdam

While Theorem 1 establishes that Decentralized Adam (DAdam) achieves the same convergence rate as centralized Adam, our empirical studies show that it often underperforms in *generalization*, especially for large-scale models such as Transformers and ResNets. Even when the training losses are similar, DAdam tends to yield slightly worse validation performance.

This discrepancy mainly arises from two intertwined effects:

1. **Amplified gradient variance due to data partitioning.** In decentralized training, each worker only observes a local subset of the global batch. Consequently, its stochastic gradient g_i is a noisier estimate of the true gradient, and the aggregate variance across workers grows with the number of participants. These noisy updates not only slow down convergence but also magnify consensus errors between neighboring models.
2. **Drifted momentum statistics.** Since each worker maintains its own local moment estimates (m_i, v_i) , the increased gradient noise causes their moving averages to diverge. This inconsistency leads to heterogeneous effective learning rates, which further destabilizes the optimization trajectory and widens the generalization gap.

Motivation A natural way to mitigate variance and stabilize momentum, and, at the same time, not to incur additional communication costs, is to *accumulate several local gradients* before updating the model—similar to gradient accumulation or large-batch training. However, naive accumulation reduces the number of model updates by a factor of s , which alters the optimization dynamics and slows training.

Algorithm 2 Decentralized Accumulated Adam on worker i (AccumAdam)

```

1:  $s \leftarrow \#$  of iterations in one accumulation loop (4, for example);  $T$  such that
   ( $T \bmod s$ ) = 0
2:  $\hat{m}_i^{(0)}, \hat{v}_i^{(0)} \leftarrow 0$ ;  $x_i^{(0)} \leftarrow x^0$ ;  $b_i \leftarrow 0$ 
3: for  $t = 1, 2, \dots, T$  do
4:    $\hat{t} \leftarrow \lceil t/s \rceil$ 
5:    $g_i^{(t)} \leftarrow \nabla \ell(x_i^{(t-1)}; \xi_i^{(t)})$ 
6:    $m_i^{(t)} \leftarrow \beta_1 \hat{m}_i^{(\hat{t}-1)} + (1 - \beta_1) g_i^{(t)}$ 
7:    $v_i^{(t)} \leftarrow \beta_2 \hat{v}_i^{(\hat{t}-1)} + (1 - \beta_2) [g_i^{(t)}]^2$ 
8:    $x_i^{(t)} \leftarrow -\alpha \frac{m_i^{(t)}/(1-\beta_1^{\hat{t}})}{\sqrt{v_i^{(t)}/(1-\beta_2^{\hat{t}})+\epsilon}} + \sum_{j \in \mathcal{N}_i^{(t)}} w_{ij}^{(t)} x_j^{(t-1)}$ 
9:    $b_i \leftarrow b_i + \frac{1}{s} g_i^{(t)}$ 
10:  if  $t \bmod s == 0$  then
11:     $\hat{m}_i^{(\hat{t})} \leftarrow \beta_1 \hat{m}_i^{(\hat{t}-1)} + (1 - \beta_1) b_i$ 
12:     $\hat{v}_i^{(\hat{t})} \leftarrow \beta_1 \hat{v}_i^{(\hat{t}-1)} + (1 - \beta_1) [b_i]^2$ 
13:     $b_i \leftarrow 0$ 
14:  end if
15: end for
    
```

Interpretation We divide the iteration indices into groups of size s :

$$t \in \{(\hat{t} - 1)s + 1, \dots, \hat{t}s\}, \quad \hat{t} = 1, 2, \dots, T/s.$$

Within each group, the local gradients are accumulated and smoothed through exponential averaging. Consequently, the effective momentum at step t can be expressed as

$$m_i^{(t)} = (1 - \beta_1) \left(\beta_1^0 g_i^{(t)} + \beta_1^1 G_i^{(\hat{t}-1)} + \dots + \beta_1^{\hat{t}-1} G_i^{(1)} \right), \quad (5.2.1)$$

where $G_i^{(\hat{t})} = \frac{1}{s} \sum_{r=(\hat{t}-1)s+1}^{\hat{t}s} g_i^{(r)}$ represents the averaged gradient within the \hat{t} -th accumulation window.

This mechanism effectively blends the short-term gradient history into each update: it smooths stochastic noise while maintaining the same update frequency T , unlike naive gradient accumulation which would reduce the number of updates to T/s .

Variance analysis and intuition Let the global mini-batch size in centralized training be B , whose gradient variance is Σ^2 . When distributed across N workers (each with a local mini-batch of size B/N), the variance of each local gradient increases to approximately $N\Sigma^2$, exacerbating consensus errors among workers.

5.2.4 Numerical experiments

Under the same settings as in Section 4.4, we present the generalization performance of decentralized training the following tables. Table 5.2.1 presents the generalization performance of All-Reduce training with Adam, decentralized Adam, and AccumAdam for the neural machine translation task with transformer [31] on WMT14 [76]. Table 5.2.2 presents the corresponding generalization performance on the image classification task on ImageNet-1K [78].

Compared with decentralized Adam, AccumAdam can achieve better performance on all metrics on both tasks, and AccumAdam can achieve comparable performance as All-Reduce training, which demonstrates its practicality.

		BLEU \uparrow / METEOR \uparrow	
Method	Topology	En-De / Transformer (base)	En-Fr / Transformer (big)
All-Reduce	Complete	27.79 \pm 0.16 / 0.5465 \pm 0.0018	45.70 \pm 0.30 / 0.6676 \pm 0.0017
DAdam	AER	27.58 \pm 0.06 / 0.5440 \pm 0.0013	45.60 \pm 0.12 / 0.6664 \pm 0.0014
	Complete	27.60 \pm 0.18 / 0.5445 \pm 0.0021	45.65 \pm 0.22 / 0.6670 \pm 0.0017
AccumAdam	AER	27.71 \pm 0.09 / 0.5481 \pm 0.0027	46.05 \pm 0.03 / 0.6715 \pm 0.0029
	Complete	27.80 \pm 0.34 / 0.5474 \pm 0.0027	46.18 \pm 0.20 / 0.6704 \pm 0.0017

Table 5.2.1: Neural Machine Translation on WMT14 [76]: The results show the generalization performance of training transformer (base) on English-to-German and transformer (big) on English-to-French. The last checkpoint of each method is evaluated by BLEU score [68] and METEOR [69] the corresponding newtest2014 testset. The number of workers is 16 and the number of workers per node is 4 for all experiments. The error bands with $\pm 2\sigma$ based on 3 runs are reported.

Methods	Top-1 Acc. \uparrow / Top-5 Acc. \uparrow / Train Loss \downarrow	
	Alternating Exp Ring	Complete
All-Reduce	N/A	76.01 / 93.05 / 1.864
DAdam	75.31 / 92.54 / 1.998	75.27 / 92.60 / 1.993
AccumAdam	76.54 / 93.18 / 1.836	76.34 / 93.15 / 1.835

Table 5.2.2: Image Classification on ImageNet-1K [78]: The results are show the generalization of training ResNet-50 on ImageNet-1K for image classification. Top-1 and Top-5 accuracies and averaged training loss in the last epoch are reported, and the accuracies are evaluated on the ImageNet validation set. The number of workers is 32 and the number of workers per node is 8 for all experiments.

5.3 Exploit consensus errors as curvature regularizer

In this section, we use the experiment of training a wide ResNet (WRN28-10) [83] on CIFAR-10 [84] as a showcase to demonstrate the limitation of DSGD and the improvement brought by our proposed algorithm. In the experiment, we employ the standard cosine annealing learning rate schedule [85] with a linear warm-up during the first 10 epochs (Figure 5.3.1, left). This learning rate schedule is commonly used

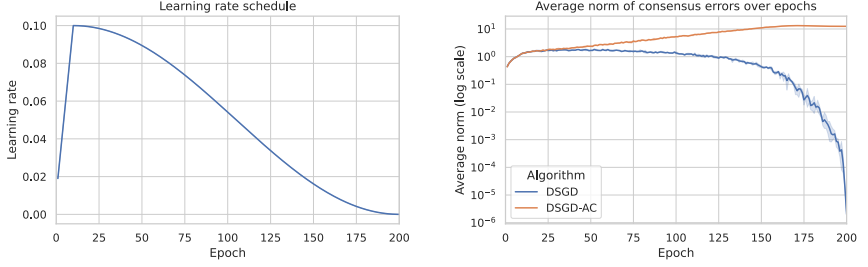


Figure 5.3.1: Decentralized training of WRN28-10 on CIFAR-10 (3 random runs for each algorithm) with 8 workers, and the communication topology is the one-peer ring topology. **Left:** Learning rate schedule (same for both algorithms). **Right:** Average norm of consensus errors evaluated at the end of every epoch ($\frac{1}{N} \sum_{i=1}^N \|x_i^{(eT)} - \bar{x}^{(eT)}\|$). p is set to 3 for DSGD-AC.

in practice and can strike a balance between the training stability and generalization [86, 87]. For decentralized training, we use 8 workers and the one-peer ring as the decentralized communication topology.

5.3.1 Vanishing potential regularizer in DSGD

We start by empirically investigating the dynamics of consensus errors when trained with DSGD. We track the average norm of the consensus errors during the training. We observe that, for DSGD, the consensus errors gradually vanish as the learning rate decreases (Figure 5.3.1, right).

From the theoretical perspective, by interpreting the mixing step as a gradient step on a quadratic consensus penalty, one obtains the per-step surrogate

$$\begin{aligned}
 J^{(t)}(x_1, \dots, x_n) &= \sum_{i=1}^n f_i(x_i^{(t)}) + \frac{1}{2\alpha^{(t)}} \sum_{i,j \in [n]} W_{ij} \|x_i^{(t)} - x_j^{(t)}\|^2 \\
 &= \underbrace{\sum_{i=1}^n f_i(\bar{x}^{(t)})}_{\text{objective on deployed model}} + \underbrace{\sum_{i=1}^n [f_i(x_i^{(t)}) - f_i(\bar{x}^{(t)})]}_{\text{sharpness}} + \underbrace{\frac{1}{2\alpha^{(t)}} \sum_{i,j \in [n]} W_{ij} \|x_i^{(t)} - x_j^{(t)}\|^2}_{\text{consensus regularizer}}
 \end{aligned} \tag{5.3.2}$$

With symmetric mixing weights and no momentum or adaptivity, each DSGD step is exactly a (stochastic) gradient on J . Thus, when $\alpha^{(t)}$ goes to 0, the consensus regularizer dominates the objective function, which minimizes the consensus errors. If considering this surrogate function, the empirical observation is not surprising because it reflects the hard constraint in the optimization problem in Eq. (2.3.1). However, the vanishing consensus errors void the sharpness term in Eq. (5.3.2) because the sharpness term because $f_i(x_i^{(t)}) \approx f_i(\bar{x}^{(t)})$ as $x_i^{(t)} - \bar{x} \rightarrow 0$. The only term left that is relevant to the deployed model $\bar{x}^{(t)}$ is the first term, which is the same objective as in synchronous SGD. Therefore, to maintain the potential benefits of free

sharpness-aware regularization [26] by the consensus errors, we need to maintain a non-vanishing radius throughout the training.

5.3.2 Algorithm: DSGD-AC (DSGD with adaptive consensus)

The proposed algorithm is shown in Algorithm 3. The difference from DSGD is highlighted, and, compared with DSGD, the proposed variant includes an adaptive factor to maintain non-diminishing consensus errors intentionally. At the end of training, the algorithm takes the global average of all local models as the deployed model.

Algorithm 3 DSGD with adaptive consensus (DSGD-AC) on worker i

Input: Dataset D , number of workers N , number of epochs E , number of batches per epoch T , initialization $x^{(0)}$, and a hyperparameter $p \in \mathbb{R}^+$.

Output: Deployed model $\bar{x} = \frac{1}{n} \sum_{j=1}^n x_j^{(TE)}$.

- 1: $x_1^{(0)} = x_2^{(0)} = \dots = x_n^{(0)} = x^{(0)}$
 - 2: **for** $t = 1$ to TE **do**
 - 3: $g_i^{(t)} = \nabla f(x_i^{(t-1)}; s_i^{(t)})$
 - 4: $\gamma^{(t)} = [\alpha^{(t)} / \alpha_{\max}]^p$
 - 5: $x_i^{(t)} = x_i^{(t-1)} - \alpha^{(t)} g_i^{(t)} + \gamma^{(t)} \sum_{j \in \mathcal{N}(i)} W_{ij} (x_j^{(t-1)} - x_i^{(t-1)})$
 - 6: **end for**
-

Note that $\alpha^{(t)}$ is determined by the learning rate scheduler like cosine annealing [85], and α_{\max} is the maximal learning rate throughout the training, which ensures $\gamma^{(t)}$ is in the range $[0, 1]$.

We evaluate the performance of DSGD-AC on classical deep learning tasks in Section 5.3.6. In the numerical experiments, the results demonstrate the superior smoothness of the loss landscapes around the found local minimas and better generalization performance of DSGD-AC over DSGD and centralized SGD. We will analyze the reasons behind this by showing that DSGD-AC maintains non-diminishing and useful consensus errors in the following sections.

5.3.3 DSGD-AC maintains non-vanishing consensus errors

The motivation of DSGD-AC is to maintain non-diminishing consensus errors. By multiplying the weight of the consensus regularizer in Eq. (5.3.2) by an adaptive γ ($\gamma \in (0, 1]$), the convergence of the consensus should be slowed down, and the idea directly gives the DSGD-AC algorithm. The per-step surrogate function of DSGD-AC is mostly the same as that of DSGD. Only the weight of the consensus regularizer becomes $\gamma^{(t)} / (2N\alpha^{(t)})$.

In this section, we investigate the impact of p on the magnitude of consensus errors. First, we can rewrite the update of DSGD-AC in matrix form,

$$X^{(t)} = X^{(t-1)} - \alpha^{(t)}G^{(t)} - \gamma^{(t)}X^{(t-1)}(I - W) = X^{(t-1)}(I - \gamma^{(t)}L) - \alpha^{(t)}G^{(t)} \quad (5.3.3)$$

where we denote the Laplacian matrix L by $L = I - W$.

By subtracting $\bar{X}^{(t)}$ on both sides of Eq. (5.3.3) and using the fact that $\Delta^{(t)} = X^{(t)}(I - \frac{1}{n}\mathbf{1}\mathbf{1}^\top)$, the dynamics of consensus errors $\Delta^{(t)}$ can be derived as

$$\begin{aligned} \Delta^{(t)} &= X^{(t-1)}(I - \gamma^{(t)}L) - \alpha^{(t)}G^{(t)} - \bar{X}^{(t)} \\ &= X^{(t-1)}(I - \gamma^{(t)}L) - \alpha^{(t)}G^{(t)} - \bar{X}^{(t-1)} + \alpha^{(t)}G^{(t)} \cdot \frac{1}{n}\mathbf{1}\mathbf{1}^\top \\ &= \Delta^{(t-1)}(I - \gamma^{(t)}L) - \alpha^{(t)}G^{(t)}(I - \frac{1}{n}\mathbf{1}\mathbf{1}^\top) \end{aligned} \quad (5.3.4)$$

Next, we denote $P = I - \frac{1}{n}\mathbf{1}\mathbf{1}^\top$ and perform an eigen-decomposition of L as $L = U_L \Lambda_L U_L^\top$. By multiply Eq. (5.3.4) by U_L from the right, we obtain

$$\begin{aligned} \Delta^{(t)}U_L &= \Delta^{(t-1)}(I - \gamma^{(t)}U_L \Lambda_L U_L^\top)U_L - \alpha^{(t)}G^{(t)}P U_L \\ &= \Delta^{(t-1)}U_L(I - \gamma^{(t)}\Lambda_L) - \alpha^{(t)}G^{(t)}P U_L \end{aligned} \quad (5.3.5)$$

By introducing $Z^{(t)} = \Delta^{(t)}U_L$ and $\tilde{G}^{(t)} = G^{(t)}P U_L$, we can re-write the update as

$$Z^{(t)} = Z^{(t-1)}(I - \gamma^{(t)}\Lambda_L) - \alpha^{(t)}\tilde{G}^{(t-1)} \quad (5.3.6)$$

Here, $Z^{(t)}$ collects the consensus error expressed in the eigenbasis of the Laplacian. The k -th row $Z_k^{(t)}$ contains the coefficients of $\Delta^{(t)}$ along the k -th Laplacian eigenvector, or *network mode*, and thus describes a characteristic pattern of disagreement across agents induced by the communication graph. We measure the overall amount of disagreement by the *disagreement radius*

$$r_t^2 := \mathbb{E} [\|\Delta^{(t)}\|_F^2].$$

Since U_L is orthogonal, $\|\Delta^{(t)}\|_F = \|Z^{(t)}\|_F$, so the radius can be equivalently studied through the Laplacian-mode dynamics of $Z^{(t)}$ in (5.3.6). By analyzing these dynamics, we obtain the following proposition.

Proposition 1 (Disagreement radius and the role of γ) *In a quasi-stationary regime with mild bounded-moment and spectral assumptions, the disagreement radius satisfies*

$$r_t^2 = \Theta \left(\frac{(\alpha^{(t)})^2}{\gamma^{(t)}} \right)$$

In particular, if $\alpha^{(t)} \rightarrow 0$ and $\gamma^{(t)}$ is bounded away from zero, then $r_t^2 \rightarrow 0$. Thus, no constant $\gamma^{(t)}$ can maintain a non-vanishing disagreement radius under diminishing stepsizes. However, if we choose $\gamma^{(t)} = g_0 (\alpha^{(t)})^p$ for some $g_0 > 0$ and $p > 0$, then as $\alpha^{(t)} \rightarrow 0$,

$$p = 2 \Rightarrow r_t^2 = \Theta(1),$$

and $p \geq 2$ is necessary to keep the consensus errors at a nontrivial scale.

Proof of Proposition 1

Recall that

$$\begin{aligned}
 P &= I - \frac{1}{n} \mathbf{1}\mathbf{1}^\top \\
 L &= I - W = U_L \Lambda_L U_L^\top \\
 \tilde{G}^{(t)} &= G^{(t)} P U_L \\
 Z^{(t)} &= Z^{(t-1)} (I - \gamma^{(t)} \Lambda_L) - \alpha^{(t)} \tilde{G}^{(t)}
 \end{aligned} \tag{5.3.7}$$

where $Z^{(t)}$ describes the consensus error projected onto the eigenbasis of the Laplacian.

Each column $z_k^{(t)}$ of $Z^{(t)}$ evolves as

$$z_k^{(t)} = (1 - \gamma^{(t)} \lambda_k(L)) z_k^{(t-1)} - \alpha^{(t)} \tilde{g}_k^{(t)} \tag{5.3.8}$$

where $\lambda_k(L)$ is the k -th eigenvalue of L .

To quantify the dynamics of $\|z_k^{(t)}\|_2^2$, consider a quasi-stationary regime where

$$\mathbb{E}[\tilde{g}_i^{(t)}] = \mu_i, \quad \mathbb{E}[\|\tilde{g}_i^{(t)} - \mu_i\|_2^2] = \sigma_i^2 \tag{5.3.9}$$

Then, by taking the expectation on Eq. (5.3.8) and letting $m_i = \mathbb{E}[z_i]$, we have

$$m_i = (1 - \gamma^{(t)} \lambda_i(L)) m_i - \alpha^{(t)} \mu_i \tag{5.3.10}$$

from which we find (for all modes $i \geq 2$)

$$m_i = -\frac{1}{\lambda_i(L)} \frac{\alpha^{(t)}}{\gamma^{(t)}} \mu_i \tag{5.3.11}$$

Next, we define $\tilde{z}_i^{(t)} = z_i^{(t)} - m_i$ so that

$$\tilde{z}_i^{(t)} = (1 - \gamma^{(t)} \lambda_i(L)) \tilde{z}_i^{(t-1)} - \alpha^{(t)} (\tilde{g}_i^{(t-1)} - \mu_i)$$

where we have subtracted m_i from both sides and used the expression for m_i just derived.

Letting $V_i = \mathbb{E}[\|\tilde{z}_i^{(t)}\|_2^2]$ and assuming that the innovation $\eta^{(t-1)} = \tilde{g}_i^{(t-1)} - \mu_i$ is conditionally independent given all events up to iteration $t-1$, we obtain

$$V_i = (1 - \gamma^{(t)} \lambda_i(L))^2 V_i + (\alpha^{(t)})^2 \sigma_i^2.$$

Solving for V_i gives

$$V_i = \frac{(\alpha^{(t)})^2}{1 - (1 - \gamma^{(t)} \lambda_i(L))^2} \sigma_i^2 = \frac{(\alpha^{(t)})^2}{2\lambda_i(L)\gamma^{(t)} - \lambda_i(L)^2(\gamma^{(t)})^2} \sigma_i^2.$$

For t large enough we have $\lambda_i(L)\gamma^{(t)} \leq 1$, so

$$\lambda_i(L)\gamma^{(t)} \leq 2\lambda_i(L)\gamma^{(t)} - \lambda_i(L)^2(\gamma^{(t)})^2 \leq 2\lambda_i(L)\gamma^{(t)}.$$

Consequently,

$$\frac{(\alpha^{(t)})^2}{2\lambda_i(L)\gamma^{(t)}} \sigma_i^2 \leq V_i \leq \frac{(\alpha^{(t)})^2}{\lambda_i(L)\gamma^{(t)}} \sigma_i^2, \quad (5.3.12)$$

so in the quasi-stationary regime we have $V_i = \Theta((\alpha^{(t)})^2/\gamma^{(t)})$. Combining (5.3.12) with (5.3.11) yields

$$\mathbb{E} \left[\|z_i^{(t)}\|_2^2 \right] = V_i + \|m_i\|_2^2 = V_i + \frac{(\alpha^{(t)})^2}{\lambda_i^2(\gamma^{(t)})^2} \|\mu_i\|_2^2.$$

If $\gamma^{(t)} \geq \gamma_{\min} > 0$ and $\alpha^{(t)} \rightarrow 0$, then by (5.3.12) we have $V_i \leq (\alpha^{(t)})^2 \sigma_i^2 / (\lambda_i(L)\gamma_{\min}) \rightarrow 0$, and similarly $\|m_i\|_2^2 = O((\alpha^{(t)})^2) \rightarrow 0$. Hence $\mathbb{E} \left[\|z_i^{(t)}\|_2^2 \right] \rightarrow 0$ for each i , so a constant consensus weight $\gamma^{(t)}$ cannot maintain a non-vanishing disagreement radius.

With the schedule $\gamma^{(t)} = g_0(\alpha^{(t)})^p$ and $g_0 > 0$, on the other hand, we obtain the lower bound

$$\mathbb{E} \left[\|z_i^{(t)}\|_2^2 \right] \geq \frac{1}{\lambda_i(L)^2 g_0^2} (\alpha^{(t)})^{2-2p} \|\mu_i\|_2^2 + \frac{1}{2\lambda_i(L)g_0} (\alpha^{(t)})^{2-p} \sigma_i^2.$$

If $p \geq 2$, at least one of the exponents $2-2p$ or $2-p$ is non-positive, so the right-hand side does not converge to zero as $\alpha^{(t)} \rightarrow 0$. Thus for $p \geq 2$ the per-mode energy

$\mathbb{E} \left[\|z_i^{(t)}\|_2^2 \right]$ is non-vanishing in the quasi-stationary regime. Finally, since

$\|Z^{(t)}\|_F^2 = \|\Delta^{(t)}\|_F^2$ by orthogonality of U , a non-vanishing $\|Z^{(t)}\|_F^2$ implies that $r_t^2 = \mathbb{E} \|\Delta^{(t)}\|_F^2$ is non-vanishing as well. \square

Interpretation of Proposition 1

The proposition establishes that DSGD-AC maintains a nontrivial level of consensus errors throughout iterations in a quasi-stationary regime. In fact, the proof of the proposition shows that the effective disagreement radius $r_t^2 = \mathbb{E} \|\Delta^{(t)}\|_F^2$ is on the order of $(\alpha^{(t)})^2/\gamma^{(t)}$. Since it has been empirically observed (see, e.g., [61, 88]) that it is advantageous to increase the radius slightly towards the end of the training, we chose $\gamma^{(t)} = g_0(\alpha^{(t)})^p$ with $p = 3$. Under cosine learning rate schedules and finite training iterations, this choice induces a mild uptick in the radius toward the final stages of training, as illustrated in Figure 5.3.1 (right). A sensitivity analysis over p in Figure 5.3.8 further supports the theory, demonstrating radius shrinkage for $p < 2$ and growth for $p > 2$ as $\alpha^{(t)} \rightarrow 0$.

5.3.4 Intrinsic property of consensus errors

Even though DSGD-AC maintains non-vanishing consensus errors, its role in leading to flatter minima and better generalization remains underexplored. In [26, Theorem 1], consensus errors are interpreted as random perturbations within the subspace

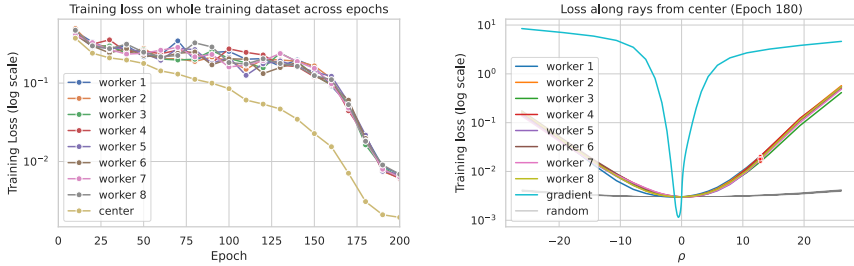


Figure 5.3.2: **Left:** Losses on the whole training dataset at local workers and global average. The losses are evaluated every 10 epochs. **Right:** Training loss at epoch 180 along: (1) worker i : lines connecting global average and worker i , (2) gradient: the line that aligns with the full-batch gradient at the global average and crosses the global average, and (3) random: 500 lines that cross the global average and follow random directions generated as in [61]. The x -axis means the directional magnitude of the perturbation along these directions. The losses are computed on $\sim 1/4$ of the training dataset due to computation complexity.

defined by the weight diversity matrix, and the resulting (asymptotically equivalent) average-direction SAM effect is shown to improve generalization. While this connection is insightful, the intrinsic structure of consensus errors (or the weight diversity matrix) has not been examined in detail.

To study this structure, we compare the training losses at local models with those at their global average. As shown in Figure 5.3.2 (left), the global average consistently achieves lower training loss than any individual worker. To further distinguish consensus errors from random perturbations, we evaluate the training losses along the directions of consensus errors and compare them against losses along sufficiently many random directions. Figure 5.3.2 (right) shows that the random directions are almost flat, which is expected given the large parameter space ($\sim 36\text{M}$ in WRN28-10) and the low-rank structure of the Hessian [89, 90]. It is also consistent with empirical observations in [34].

In contrast, directions induced by consensus errors yield significant increases in training loss, highlighting that these errors are not random but aligned with directions of higher curvature. This phenomenon suggests a correlation between consensus errors and the dominant subspace of the Hessian (or directions with larger curvature). Motivated by this observation, we formalize it in the following proposition.

Proposition 2 (Structure of consensus errors) *Let x^* be a locally strongly convex minimizer of F with Hessian $H = U_H \Lambda_H U_H^\top$ and eigenvalues $0 < \lambda_1(H) \leq \dots \leq \lambda_d(H)$. Assume i.i.d. local objectives ($f_i \equiv F$), and let W be a symmetric, doubly stochastic communication matrix associated with a connected undirected graph. Let $\Delta^{(t)}$ denote the consensus error at time t . Consider the DSGD-AC recursion in a neighbourhood of x^* and approximate the local gradients by their first-order Taylor expansion, $\nabla f_i(x_i^{(t)}) \approx H(x_i^{(t)} - x^*)$. For non-increasing stepsizes $\alpha^{(t)} > 0$ and consensus factors $\gamma^{(t)} > 0$, the projection $\Delta_k^{(t)} := u_k^\top \Delta^{(t)}$ of the*

consensus error onto each Hessian eigenvector u_k then evolves as a scalar linear system whose stability requires

$$\alpha^{(t)} < \frac{2 + (\lambda_{\min}(W) - 1)\gamma^{(t)}}{\lambda_k(H)}, \quad (5.3.13)$$

where $\lambda_{\min}(W)$ is the smallest eigenvalue of W . The right-hand side of (5.3.13) is decreasing in $\lambda_k(H)$, so with non-increasing stepsizes $\alpha^{(t)}$ modes corresponding to smaller eigenvalues enter the stable regime earlier, while high-curvature modes remain closer to instability for longer and therefore retain higher variance under the same injected noise.

Proof of Proposition 2

In this proof, we focus on a local minima x^* of $F(x)$ that is locally strongly convex, and we denote $X^* = [x^*, \dots, x^*] \in \mathbb{R}^{d \times n}$.

Rewrite the update of DSGD-AC by replacing the stochastic gradient with its expectation and a noise matrix,

$$X^{(t)} = X^{(t-1)}(I - \gamma^{(t)}L) - \alpha^{(t)}(\nabla F(X^{(t-1)}) + \Xi^{(t)}) \quad (5.3.14)$$

where

$$\begin{aligned} \nabla F(X^{(t-1)}) &:= [\nabla f_1(x_1), \dots, \nabla f_n(x_n)] \\ \Xi^{(t)} &:= G^{(t)} - \nabla F(X^{(t-1)}) \end{aligned} \quad (5.3.15)$$

Then, in a similar way as in the proof of Prop. 1, we can rewrite the update of $\Delta^{(t)}$ as

$$\Delta^{(t)} = \Delta^{(t-1)}(I - \gamma^{(t)}L) - \alpha^{(t)}\nabla F(X^{(t-1)})P - \alpha^{(t)}\Xi^{(t)}P \quad (5.3.16)$$

By further assuming the local data distributions are i.i.d., we have $f_i = F$ and $H_i = H$ for all $i \in [n]$. By doing Taylor's expansion on x^* and ignoring the higher-order terms, we have

$$\nabla f_i(x_i^{(t-1)}) \approx \nabla f_i(x^*) + H_i(x_i^{(t-1)} - x^*) = H(x_i^{(t-1)} - x^*) \quad (5.3.17)$$

then we have

$$\nabla F(X^{(t-1)}) \approx H(X^{(t-1)} - X^*) \Rightarrow \nabla F(X^{(t-1)})P \approx H\Delta^{(t-1)} \quad (5.3.18)$$

Therefore, by plugging Eq. (5.3.18) back to Eq. (5.3.16), we have

$$\Delta^{(t)} \approx \Delta^{(t-1)}(I - \gamma^{(t)}L) - \alpha^{(t)}H\Delta^{(t-1)} - \alpha^{(t)}\Xi^{(t)}P \quad (5.3.19)$$

Take the eigen decomposition of H , we have $H = U\Lambda U^\top$. We denote the k -th eigenvector of H by u_k , the projection of consensus error on u_k by $\Delta_k^{(t)} := u_k^\top \Delta^{(t)}$, and the projection of gradient noise by $\xi_k^{(t)} = u_k^\top \Xi^{(t)}P$.

By multiplying u_k^\top on both sides of Eq. (5.3.19) from the left, we have the dynamics of the projection of the consensus errors on u_k as

$$\begin{aligned}\Delta_k^{(t)} &\approx \Delta_k^{(t-1)}(I - \gamma^{(t)}L) - \alpha^{(t)}\lambda_k\Delta_k^{(t-1)} - \alpha^{(t)}\Xi_k^{(t)} \\ &= \Delta_k^{(t-1)}(I - \gamma^{(t)}L - \alpha^{(t)}\lambda_kI) - \alpha^{(t)}\Xi_k^{(t)}\end{aligned}\quad (5.3.20)$$

which is a linear system of $\Delta_k^{(t)}$ driven by noise ξ_k .

Now we focus on the steady-state covariance of $\Delta_k^{(t)}$,

$$S_k := \lim_{t \rightarrow \infty} \mathbb{E}[\Delta_k^{(t)\top} \Delta_k^{(t)}] \quad (5.3.21)$$

which satisfies the discrete Lyapunov equation

$$S_k = A_k S_k A_k^\top + [\alpha^{(t)}]^2 \Sigma_k \quad (5.3.22)$$

where $A_k = I - \gamma^{(t)}L - \alpha^{(t)}\lambda_kI$ and Σ_k is the covariance matrix of the projected noise.

We denote the eigenvalues of W by $\{\lambda_1^W, \dots, \lambda_n^W\}$. The system is stable only if the eigenvalues of A_k are all in $(-1, 1)$, from which we can derive the condition on $\alpha^{(t)}$ by

$$-1 < 1 - \gamma^{(t)} + \gamma^{(t)}\lambda_j^W - \alpha^{(t)}\lambda_k < 1 \quad \forall j \in [n] \quad (5.3.23)$$

By using the fact that W is a non-negative and doubly stochastic matrix which defines a communication topology, we have $\lambda_j^W \in (-1, 1]$ for all $j \in [n]$. Recall that x^* is a local minima of $F(x)$ that is locally strongly convex, we have $\lambda_k > 0$. Then we can derive the condition on $\alpha^{(t)}$ that makes the system stable, which is

$$\alpha^{(t)} < \frac{2 + (\lambda_{\min}^W - 1)\gamma^{(t)}}{\lambda_k} \quad (5.3.24)$$

From Eq. (5.3.24), we can derive that, for the same set of α and γ , the condition is easier to fulfill with smaller λ_k . Therefore, the system is stable, or the contraction of S_k happens earlier, with a smaller eigenvalue λ_k and with non-increasing step sizes.

With the contractions happening on the subspace spanned by eigenvectors with small eigenvalues, the consensus errors align with the eigenvectors that have larger eigenvalues, which is the dominant subspace of the Hessian. \square

Remark 1 (*Theoretical benefit of adaptive consensus*) A smaller consensus factor $\gamma^{(t)}$ relaxes the stability condition in (5.3.13). As $\gamma^{(t)}$ decreases during training, more low-curvature modes become stable, while the high-curvature modes remain closer to instability. As a result, the consensus errors gradually concentrate on a lower-dimensional subspace spanned by the dominant Hessian directions.

Remark 2 (*Alignment is meaningful only with a controlled disagreement radius*)

The conclusion of Proposition 2 only holds when the disagreement radius stays in a reasonable range. Although one may relax the condition (5.3.13) by the selection of W and $\gamma^{(t)}$, taking $\gamma^{(t)}$ too small or using a graph with a very large $\lambda_{\min}(W)$ can cause the disagreement to grow quickly (Proposition 1). In that case, the iterates may move out of the region where the local Taylor approximation is accurate. As shown in Figure 5.3.2 (right), the losses along the line passing the global average and the worker model can be well approximated by a quadratic function, which validates the local Taylor approximation in the experiments. On the other hand, if the radius becomes too small, the disagreement barely perturbs the model, and its directional structure becomes unimportant. Thus, the alignment effect is useful only when the disagreement radius is neither too large nor too small.

Limitation While the alignment exists and can be shown theoretically, the alignment is noisy and spans on less-sharp directions when compared with the gradient direction. As shown in Figure 5.3.2 (right), the gradient computed on the corresponding dataset leads to a sharper increase than the consensus errors. An interesting direction for future work could be an improved algorithm based on DSGD-AC that can utilize the gradient information to promote the concentration of consensus errors on the dominant Hessian subspace with small computational overhead.

5.3.5 Consensus error as a curvature regularizer

In this section we relate the consensus errors maintained by DSGD-AC to the local geometry of the global objective. Our goal is to understand which perturbations of the deployed model $\bar{x}(t)$ are implicitly favored or suppressed by the algorithm.

Lemma 1 shows that, in a neighbourhood of a locally strongly convex minimizer x^* , the average local loss decomposes into the loss at the deployed model plus a quadratic envelope term depending on the disagreement covariance Σ_t , up to higher-order corrections. Thus, in this regime, DSGD-AC can be viewed as minimizing $F(\bar{x}(t))$ together with a Hessian-weighted disagreement penalty.

Lemma 1 (Local loss envelop) *Let x^* be a locally strongly convex minimizer of F with Hessian H at x^* . For a fixed time t , let $\bar{x}(t) := \frac{1}{n} \sum_{i=1}^n x_i^{(t)}$ be the deployed model and define the disagreements $e_i^{(t)} := x_i^{(t)} - \bar{x}(t)$. Let*

$$\Sigma_t := \frac{1}{n} \sum_{i=1}^n e_i^{(t)} e_i^{(t)\top}$$

denote their empirical covariance. Assume that $\|x_i^{(t)} - x^\|$ is small for all i , then*

$$\frac{1}{n} \sum_{i=1}^n f_i(x_i^{(t)}) = F(\bar{x}(t)) + \frac{1}{2} \text{tr}(H\Sigma_t) + O((\text{tr}\Sigma_t)^{3/2}). \quad (5.3.25)$$

Proof of Lemma 1

Fix t and abbreviate $\bar{x} = \bar{x}^{(t)}$, $e_i = e_i^{(t)}$. A Taylor expansion of f_i around \bar{x} yields

$$f_i(\bar{x} + e_i) = f_i(\bar{x}) + \nabla f_i(\bar{x})^\top e_i + \frac{1}{2} e_i^\top H e_i + R_i,$$

where $R_i = O(\|e_i\|^3)$ and we used that all f_i have Hessian H at x^* . Averaging over i gives

$$\frac{1}{n} \sum_{i=1}^n f_i(x_i^{(t)}) = F(\bar{x}) + \frac{1}{n} \sum_{i=1}^n \nabla f_i(\bar{x})^\top e_i + \frac{1}{2n} \sum_{i=1}^n e_i^\top H e_i + \frac{1}{n} \sum_{i=1}^n R_i.$$

By definition of \bar{x} we have $\sum_{i=1}^n \delta_i = 0$, hence

$$\frac{1}{n} \sum_{i=1}^n \nabla f_i(\bar{x})^\top e_i = \nabla F(\bar{x})^\top \left(\frac{1}{n} \sum_{i=1}^n e_i \right) = 0.$$

Moreover,

$$\frac{1}{2n} \sum_{i=1}^n e_i^\top H e_i = \frac{1}{2} \operatorname{tr} \left(H \frac{1}{n} \sum_{i=1}^n e_i e_i^\top \right) = \frac{1}{2} \operatorname{tr}(H \Sigma_t).$$

For the remainder, there exists a constant $C > 0$ such that $|R_i| \leq C \|\delta_i\|^3$ in the local region. Thus

$$\left| \frac{1}{n} \sum_{i=1}^n R_i \right| \leq \frac{C}{n} \sum_{i=1}^n \|e_i\|^3 \leq C \left(\frac{1}{n} \sum_{i=1}^n \|e_i\|^2 \right)^{3/2} = C (\operatorname{tr} \Sigma_t)^{3/2},$$

where the second step follows from Hölder's inequality. This gives the claimed $O((\operatorname{tr} \Sigma_t)^{3/2})$ bound and for the remainder and concludes the proof. \square

To understand how this disagreement penalty depends on curvature and on the communication graph, we diagonalize the local dynamics in the joint eigenbasis of the Hessian and the Laplacian. This leads to the following spectral representation.

Proposition 3 (Curvature tilt) *Under the assumptions and notation of Proposition 2 and Lemma 1, fix a time t in a local quasi-stationary regime and freeze $\alpha = \alpha^{(t)}$ and $\gamma = \gamma^{(t)}$. Let $L = I - W$ be the graph Laplacian and denote its eigenvalues by $0 = \lambda_1(L) < \lambda_2(L) \leq \dots \leq \lambda_N(L)$, and let $\lambda_1(H) \leq \dots \leq \lambda_d(H)$ be the eigenvalues of H . Let Σ_t be the disagreement covariance at time t . Then the leading-order Hessian-weighted disagreement envelope can be written as*

$$\frac{1}{2} \mathbb{E}[\operatorname{tr}(H \Sigma_t)] \approx \frac{(\alpha^{(t)})^2}{4N} \sum_{j=2}^N \sum_{k=1}^d w_j(\lambda_k(H)) q_{k,j}, \quad (5.3.26)$$

where $q_{k,j} \geq 0$ is the innovation variance of the Laplacian-Hessian mode (j, k) and, for $\lambda \geq 0$,

$$w_j(\lambda) := \frac{\lambda}{\gamma^{(t)} \lambda_j(L) + \alpha^{(t)} \lambda}. \quad (5.3.27)$$

For each fixed graph mode $j \geq 2$, the weight $w_j(\lambda)$ is strictly increasing in λ .

Proof of Proposition 3

We work in the local quadratic regime around x^* and on a short time window around t where we freeze $\alpha = \alpha^{(t)}$ and $\gamma = \gamma^{(t)}$. From the linearization in Appendix A.2 (cf. the proof of Proposition 2), we have

$$\Delta^{(s+1)} \approx \Delta^{(s)}(I - \gamma L) - \alpha H \Delta^{(s)} - \alpha \Xi^{(s+1)} P, \quad (5.3.28)$$

for s in a short window around t , where $\Xi^{(s+1)}$ collects the gradient noise and P is the projection onto the disagreement subspace.

Let $L = U_L \Lambda_L U_L^\top$ and $H = U_H \Lambda_H U_H^\top$ be the eigendecompositions of the Laplacian and Hessian, with eigenvalues $0 = \lambda_1(L) < \lambda_2(L) \leq \dots \leq \lambda_n(L)$ and $0 < \lambda_1(H) \leq \dots \leq \lambda_d(H)$. We write the consensus error in the joint eigenbasis as

$$\Delta^{(s)} = U_H Y^{(s)} U_L^\top,$$

for some coefficient matrices $Y^{(s)} \in \mathbb{R}^{d \times N}$, and define the corresponding noise coefficients

$$Z^{(s+1)} := U_H^\top \Xi^{(s+1)} P U_L.$$

Substituting these into (5.3.28) and using $I - \gamma L = U_L (I - \gamma \Lambda_L) U_L^\top$ and $H = U_H \Lambda_H U_H^\top$ gives

$$Y^{(s+1)} = Y^{(s)}(I - \gamma \Lambda_L) - \alpha \Lambda_H Y^{(s)} - \alpha Z^{(s+1)}.$$

Taking the (k, j) entry yields, for $k = 1, \dots, d$ and $j = 1, \dots, N$,

$$Y_{k,j}^{(s+1)} = a_{k,j} Y_{k,j}^{(s)} - \alpha \zeta_{k,j}^{(s+1)}, \quad a_{k,j} := 1 - \gamma \lambda_j(L) - \alpha \lambda_k(H), \quad (5.3.29)$$

where $\zeta_{k,j}^{(s+1)} := Z_{k,j}^{(s+1)}$. Since $\Delta^{(s)}$ lies in the disagreement subspace, the consensus graph mode $j = 1$ does not contribute and we may restrict to $j \geq 2$.

On the short time window around t , we approximate (5.3.29) as a stationary AR(1) recursion driven by zero-mean innovations with variance

$$q_{k,j} := \text{Var}(\zeta_{k,j}^{(s)}).$$

Assuming $|a_{k,j}| < 1$ (the stability condition of Proposition 2) and that the innovations are uncorrelated across the short time window, the stationary variance $S_{k,j} := \text{Var}(Y_{k,j})$ satisfies the scalar Lyapunov equation

$$S_{k,j} = a_{k,j}^2 S_{k,j} + \alpha^2 q_{k,j},$$

hence

$$S_{k,j} = \frac{\alpha^2}{1 - a_{k,j}^2} q_{k,j}. \quad (5.3.30)$$

Using $a_{k,j} = 1 - \gamma \lambda_j(L) - \alpha \lambda_k(H)$, we compute

$$1 - a_{k,j}^2 = 1 - (1 - \gamma \lambda_j(L) - \alpha \lambda_k(H))^2 = 2(\gamma \lambda_j(L) + \alpha \lambda_k(H)) - (\gamma \lambda_j(L) + \alpha \lambda_k(H))^2.$$

In the small-stepsize regime where $\gamma\lambda_j(L) + \alpha\lambda_k(H)$ is small, the quadratic term can be neglected and we obtain the approximation

$$S_{k,j} \approx \frac{\alpha^2}{2(\gamma\lambda_j(L) + \alpha\lambda_k(H))} q_{k,j}. \quad (5.3.31)$$

Next, recall that

$$\Sigma_t = \frac{1}{n} \mathbb{E}[\Delta^{(t)} \Delta^{(t)\top}].$$

Using $\Delta^{(t)} = U_H Y^{(t)} U_L^\top$ and orthogonality of U_H and U_L , we obtain

$$\mathbb{E}[\text{tr}(H\Sigma_t)] = \frac{1}{n} \mathbb{E}[\text{tr}(H\Delta^{(t)} \Delta^{(t)\top})] = \frac{1}{n} \mathbb{E}[\text{tr}(\Lambda_H Y^{(t)} Y^{(t)\top})].$$

The last trace equals $\sum_{k=1}^d \lambda_k(H) \sum_{j=1}^n \mathbb{E}[(Y_{k,j}^{(t)})^2]$. Approximating $\mathbb{E}[(Y_{k,j}^{(t)})^2]$ by the stationary variance $S_{k,j}$ in (5.3.31) for $j \geq 2$ and noting again that the $j = 1$ consensus mode does not contribute, we obtain

$$\mathbb{E}[\text{tr}(H\Sigma_t)] \approx \frac{\alpha^2}{2n} \sum_{j=2}^n \sum_{k=1}^d \frac{\lambda_k(H)}{\gamma\lambda_j(L) + \alpha\lambda_k(H)} q_{k,j}.$$

Multiplying by $\frac{1}{2}$ yields

$$\frac{1}{2} \mathbb{E}[\text{tr}(H\Sigma_t)] \approx \frac{\alpha^2}{4n} \sum_{j=2}^n \sum_{k=1}^d w_j(\lambda_k(H)) q_{k,j},$$

with

$$w_j(\lambda) := \frac{\lambda}{\gamma\lambda_j(L) + \alpha\lambda},$$

which is exactly (5.3.26)–(5.3.27).

Finally, for each fixed $j \geq 2$ we have $\lambda_j(L) > 0$ and $\alpha, \gamma > 0$, so for $\lambda \geq 0$,

$$w'_j(\lambda) = \frac{\gamma\lambda_j(L)}{(\gamma\lambda_j(L) + \alpha\lambda)^2} > 0.$$

Thus $w_j(\lambda)$ is strictly increasing in λ for every $j \geq 2$, which completes the proof.

The spectral form in Proposition 3 separates the envelope into curvature-dependent weights $w_j(\lambda_k(H))$ and mode-wise innovation variances $q_{k,j}$. To go further, we specialize to the case where these innovations arise from mini-batch SGD noise. A growing body of empirical and theoretical work has shown that, near a local minimum, the covariance of mini-batch SGD gradients is approximately Hessian-aligned and scales with both the loss value and curvature, $\text{Cov}(g(x) - \nabla F(x)) \approx c_t L(x) H(x)$, in linear models and deep networks (e.g., [91], [92], [93]). Under this structure the $q_{k,j}$ inherit the same dependence on the Hessian eigenvalues, which yields a curvature-dependent spectral penalty of the form in (5.3.33).

Corollary 1 (Hessian-aligned mini-batch noise) *Under the assumptions and notation of Proposition 3, assume in addition that the gradient noise driving DSGD-AC is inherited from a mini-batch SGD oracle whose covariance is approximately Hessian-aligned,*

$$\text{Cov}(g_i(x) - \nabla f_i(x)) \approx c_t L(x) H(x), \quad (5.3.32)$$

for some scalar factor $c_t > 0$ depending on the batch size and possibly on time t . Then, in the local quadratic regime around x^* , the leading-order Hessian-weighted disagreement envelope can be written as

$$\frac{1}{2} \mathbb{E}[\text{tr}(H\Sigma_t)] \approx L(\bar{x}^{(t)}) \sum_{k=1}^d \omega_t(\lambda_k(H)), \quad (5.3.33)$$

where $\omega_t : [0, \infty) \rightarrow [0, \infty)$ is strictly increasing and satisfies that $\lambda \mapsto \omega_t(\lambda)/\lambda$ is also strictly increasing on $(0, \infty)$. In particular, larger Hessian eigenvalues receive a disproportionately larger penalty relative to their magnitude than smaller ones.

Proof of Corollary 1

By Proposition 3, the leading-order envelope can be written as

$$\frac{1}{2} \mathbb{E}[\text{tr}(H\Sigma_t)] \approx \frac{(\alpha^{(t)})^2}{4n} \sum_{j=2}^n \sum_{k=1}^d w_j(\lambda_k(H)) q_{k,j},$$

with

$$w_j(\lambda) = \frac{\lambda}{\gamma^{(t)} \lambda_j(L) + \alpha^{(t)} \lambda},$$

and $q_{k,j}$ the innovation variances of the joint Laplacian–Hessian modes. Under the Hessian-aligned covariance structure (5.3.32), the per-step gradient noise covariance in the Hessian eigenbasis is approximately diagonal with entries proportional to $L(\bar{x}^{(t)}) \lambda_k(H)$. Projecting into the joint basis, the $q_{k,j}$ inherit this alignment and, up to graph-dependent constants, satisfy

$$q_{k,j} \approx c_t L(\bar{x}^{(t)}) \lambda_k(H).$$

Substituting this scaling gives

$$\frac{1}{2} \mathbb{E}[\text{tr}(H\Sigma_t)] \approx L(\bar{x}^{(t)}) \frac{(\alpha^{(t)})^2 c_t}{4N} \sum_{j=2}^N \sum_{k=1}^d \frac{\lambda_k(H)^2}{\gamma^{(t)} \lambda_j(L) + \alpha^{(t)} \lambda_k(H)}.$$

and we recover (5.3.33), with

$$\omega_t(\lambda) := \frac{(\alpha^{(t)})^2 c_t}{4N} \sum_{j=2}^N \frac{\lambda^2}{\gamma^{(t)} \lambda_j(L) + \alpha^{(t)} \lambda}.$$

It remains to verify the monotonicity properties of ω_t . For each fixed $j \geq 2$, define

$$h_{t,j}(\lambda) := \frac{\lambda^2}{\gamma^{(t)}\lambda_j(L) + \alpha^{(t)}\lambda} = \lambda w_j(\lambda),$$

where $w_j(\lambda)$ is the weight from Proposition 3. We have already shown that, for $\lambda \geq 0$, $w_j(\lambda) \geq 0$ and $w_j(\lambda)$ is strictly increasing. Therefore, for $\lambda > 0$,

$$h'_{t,j}(\lambda) = w_j(\lambda) + \lambda w'_j(\lambda) > 0,$$

so each $h_{t,j}$ is strictly increasing on $(0, \infty)$ (and nondecreasing on $[0, \infty)$). Since ω_t is a positive linear combination of the $h_{t,j}$, $\omega_t(\lambda)$ is strictly increasing on $(0, \infty)$.

Moreover,

$$\frac{\phi_t(\lambda)}{\lambda} = \frac{(\alpha^{(t)})^2 c_t}{4n} \sum_{j=2}^n \frac{\lambda}{\gamma^{(t)}\lambda_j(L) + \alpha^{(t)}\lambda} = \frac{(\alpha^{(t)})^2 c_t}{4n} \sum_{j=2}^n w_j(\lambda).$$

Each $w_j(\lambda)$ is strictly increasing in λ by Proposition 3, so their positive linear combination $\omega_t(\lambda)/\lambda$ is also strictly increasing on $(0, \infty)$. This proves the monotonicity stated in the corollary.

Remark The spectral form in (5.3.33) shows that, under Hessian-aligned mini-batch noise, the leading-order loss inflation induced by DSGD-AC behaves like an implicit spectral penalty of the form $L(\bar{x}(t)) \sum_k \omega_t(\lambda_k(H))$, where both $\omega_t(\lambda)$ and $\omega_t(\lambda)/\lambda$ are strictly increasing. In particular, larger eigenvalues of H are penalized disproportionately more per unit curvature than smaller ones. This contrasts with classical criteria that depend only on $\text{tr}(H)$ or $\log \det H$, and implies that the top eigenvalues of H are implicitly regularized by the combination of consensus noise and mini-batch SGD. Conceptually, this connects DSGD-AC to explicit eigenvalue regularization schemes that aim to control large curvature directions in sharpness-aware methods such as Eigen-SAM [94] or Hessian-based noise-stability regularization [95], but here the regularization arises automatically from decentralized averaging and stochastic gradients rather than from additional optimization steps.

5.3.6 Numerical experiments of DSGD-AC

In this section, we present the results of the numerical experiments on image classification with wide ResNet [83] and on machine translation with transformers [31]. In the experiments, we follow hyperparameters in the corresponding original papers, and we reproduce the same baseline performance for a fair comparison.

Each set of experiments consists of three random runs with fixed random seeds. We report $2 \times$ standard deviation in all tables, and the shaded areas in plots correspond to the 95% confidence interval.

5.3.7 Image classification with wide ResNet

We train two variants of Wide ResNets (WRN28-10 and WRN16-8) [83] on two datasets, CIFAR-10 and CIFAR-100 [84], and we present the curves of the classification accuracies on the test set and training/test losses in Figures 5.3.3-5.3.6. The test performance and the flatness of the solutions are reported in Table 5.3.3.

Since finding the best sharpness metric that always reflects the potential generalization is still an open question, we use the top-1 eigenvalue as a surrogate, which is widely used in other literature and proven to have a strong correlation [61, 94] with the generalization.

Model	Dataset	Algorithm	Test Acc. (%) \uparrow	Test Loss \downarrow	Mean λ_{\max} \downarrow	Compute \downarrow
WRN28-10	CIFAR-10	DSGD	96.07 \pm 0.13	0.176 \pm 0.005	22.4360 \pm 3.9916	1X
		SGD	95.96 \pm 0.14	0.182 \pm 0.004	16.8485 \pm 0.3251	1X
		DSGD-AC	<u>96.77 \pm 0.11</u>	<u>0.128 \pm 0.003</u>	<u>8.9693 \pm 0.3514</u>	1X
		AD-SAM	96.37 \pm 0.11	0.168 \pm 0.002	24.9059 \pm 1.6212	1X
		SAM	97.33 \pm 0.04	0.100 \pm 0.002	0.3523 \pm 0.0312	2X
	CIFAR-100	DSGD	79.86 \pm 0.22	0.899 \pm 0.008	49.5719 \pm 4.8022	1X
		SGD	80.15 \pm 0.42	0.878 \pm 0.020	37.3799 \pm 2.8886	1X
		DSGD-AC	<u>82.38 \pm 0.09</u>	<u>0.755 \pm 0.008</u>	<u>19.8061 \pm 0.6653</u>	1X
		AD-SAM	<u>82.57 \pm 0.31</u>	0.891 \pm 0.007	32.6371 \pm 2.3362	1X
		SAM	83.79 \pm 0.25	0.618 \pm 0.003	1.7295 \pm 0.0385	2X
WRN16-8	CIFAR-10	DSGD	95.94 \pm 0.11	0.152 \pm 0.001	18.1998 \pm 0.6427	1X
		SGD	95.81 \pm 0.13	0.153 \pm 0.003	17.4934 \pm 1.6191	1X
		DSGD-AC	<u>96.17 \pm 0.04</u>	<u>0.129 \pm 0.003</u>	<u>11.8250 \pm 0.4883</u>	1X
		AD-SAM	<u>96.25 \pm 0.12</u>	0.152 \pm 0.002	<u>8.5178 \pm 0.5453</u>	1X
		SAM	96.81 \pm 0.08	0.102 \pm 0.003	1.3928 \pm 0.0586	2X
	CIFAR-100	DSGD	79.25 \pm 0.26	0.854 \pm 0.016	36.1998 \pm 3.8028	1X
		SGD	79.42 \pm 0.18	0.849 \pm 0.015	33.7733 \pm 0.7897	1X
		DSGD-AC	80.67 \pm 0.11	<u>0.771 \pm 0.005</u>	19.8032 \pm 0.1652	1X
		AD-SAM	<u>81.36 \pm 0.06</u>	0.858 \pm 0.004	<u>17.5450 \pm 1.2583</u>	1X
		SAM	81.51 \pm 0.08	0.677 \pm 0.003	4.7932 \pm 0.1957	2X

Table 5.3.3: Algorithm comparison on image classification. We use 8 workers and one-peer ring topology for decentralized methods. The best in each experiment setup is **bold**, and the second best is underlined.

CHAPTER 5. ROBUST DECENTRALIZED TRAINING

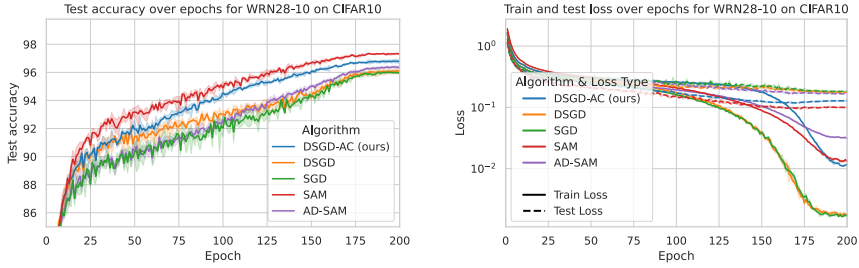


Figure 5.3.3: WRN28-10 on CIFAR-10. **Left:** Test accuracy on test set. For decentralized training, the accuracy is evaluated on the global average model. **Right:** Training and test losses.

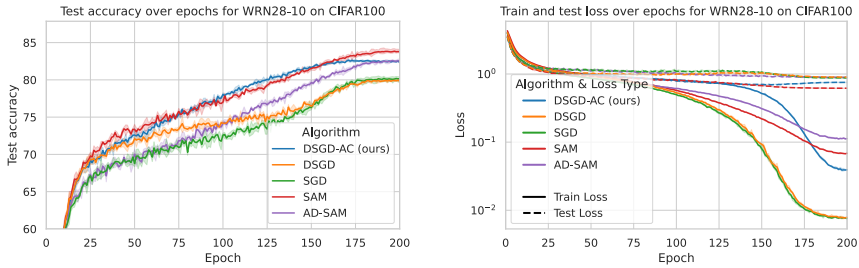


Figure 5.3.4: WRN28-10 on CIFAR-100. **Left:** Test accuracy on test set. For decentralized training, the accuracy is evaluated on the global average model. **Right:** Training and test losses.

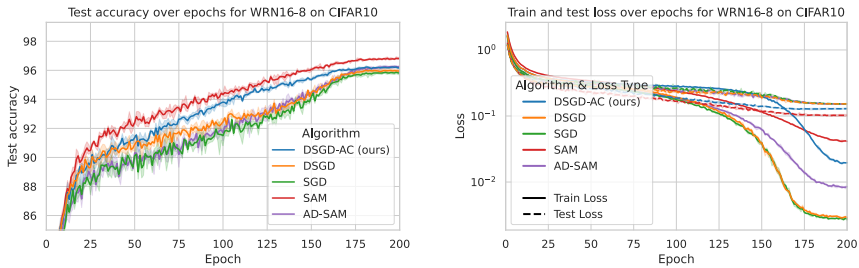


Figure 5.3.5: WRN16-8 on CIFAR-10. **Left:** Test accuracy on test set. For decentralized training, the accuracy is evaluated on the global average model. **Right:** Training and test losses.

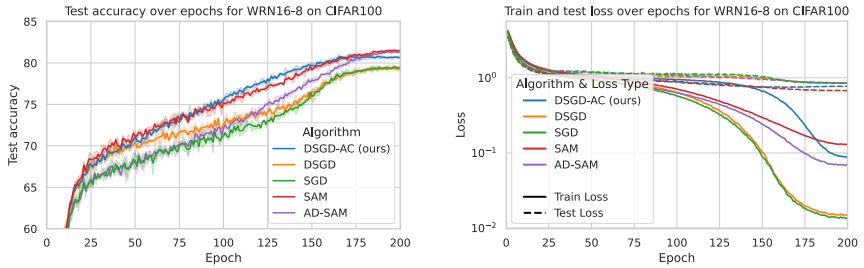


Figure 5.3.6: WRN16-8 on CIFAR-100. **Left:** Test accuracy on test set. For decentralized training, the accuracy is evaluated on the global average model. **Right:** Training and test losses.

In the experiment results, DSGD-AC outperforms DSGD and SGD in test accuracy, test losses, and solution flatness by a clear margin. Moreover, it can also be seen that DSGD can not outperform SGD with its best performance.

Sensitivity analysis of p

In all experiments, we use $p = 3$ for DSGD-AC, which is based on experiment tuning. The test results with $p = \{0, 1, 2, 3, 4, 5\}$ are presented in Figure 5.3.7 and Table 5.3.4. The tracked average norm of consensus errors with varying p is shown in Figure 5.3.8.

Note that DSGD-AC with $p = 0$ is equivalent to DSGD. The results demonstrate the effectiveness of introducing p and DSGD-AC, and $p = 3$ brings the best performance.

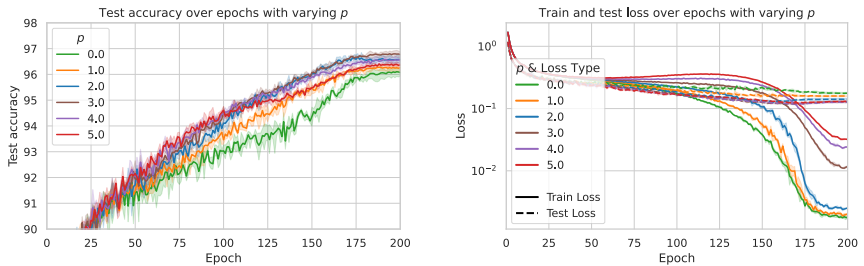


Figure 5.3.7: DSGD(-AC) on WRN28-10 on CIFAR-10 with varying p . **Left:** Test accuracy on test set. For decentralized training, the accuracy is evaluated on the global average model. **Right:** Training and test losses.

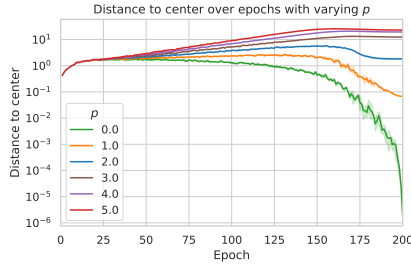


Figure 5.3.8: Average norm of consensus errors over epochs with varying p .

p	Test Accuracy (%) \uparrow	Train Loss \downarrow	Test Loss \downarrow
0	96.07 \pm 0.13	0.002 \pm 0.000	0.176 \pm 0.005
1	96.26 \pm 0.14	0.002 \pm 0.000	0.159 \pm 0.003
2	<u>96.58</u> \pm 0.18	0.003 \pm 0.000	0.141 \pm 0.006
3	96.77 \pm 0.11	0.012 \pm 0.000	<u>0.128</u> \pm 0.003
4	<u>96.53</u> \pm 0.13	0.024 \pm 0.001	0.127 \pm 0.004
5	96.37 \pm 0.04	0.032 \pm 0.001	0.130 \pm 0.002

Table 5.3.4: Sensitivity analysis of parameter p in the WRN28-10 on CIFAR10 experiment. The best value is **bold**, and the second best is underlined.

5.3.8 Machine translation with transformers

We also validate the idea of controlling consensus errors on transformer models by simply replacing the local update with the Adam optimizer [81]. DSGD-AC is then adapted to DAdam-AC. We train Transformer (the big variant, ~ 213 M parameters) [31] on WMT14 (English-to-German) [76] and present the curves of training losses and BLEU scores on the test set. The BLEU scores [68] (which is used to evaluate the translation quality in the original paper) and the losses on the test set and the training set are reported in Table 5.3.5.

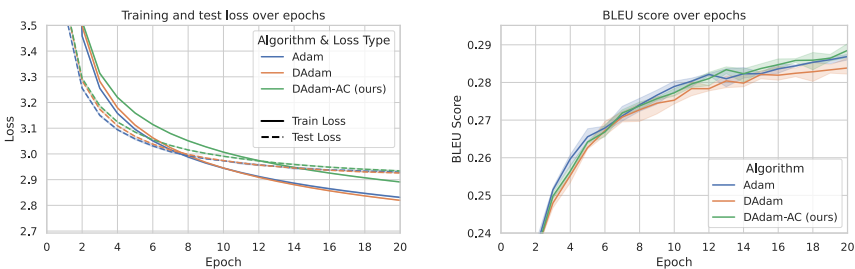


Figure 5.3.9: Transformer (big) on WMT14 English-to-German. **Left**: Losses on training set. **Right**: BLEU scores on the test set.

Algorithm	BLEU score \uparrow	Test loss \downarrow	Train loss \downarrow
Adam	28.68 \pm 0.07	2.9290 \pm 0.0026	2.8310 \pm 0.0019
DAdam	28.38 \pm 0.22	2.9258 \pm 0.0018	2.8195 \pm 0.0008
DAdam-AC	28.85 \pm 0.18	2.9338 \pm 0.0017	2.8909 \pm 0.0012

Table 5.3.5: Performance comparison of DAdam, Adam, and DAdam-AC on neural machine translation with the transformer model.

The results demonstrate that DAdam-AC can outperform other baselines on the translation quality metric. The adaptive consensus brings substantial improvement compared with DAdam. We believe further improvement is possible if we take the adaptive consensus into account when designing the optimizer.

5.4 Summary

This chapter revisited the relationship between decentralization and generalization in distributed deep learning, questioning the long-held view that decentralized training should simply replicate centralized optimization. Through the study of *Decentralized Adam (DAdam)* and *AccumAdam*, we showed that while both achieve convergence rates comparable to centralized Adam, differences in local gradient statistics can lead to distinct generalization behavior. AccumAdam mitigates this effect by accumulating gradients across steps, narrowing the generalization gap while preserving scalability. We then explored whether consensus errors—typically treated as undesirable—could instead enhance learning. By introducing *DSGD with Adaptive Consensus (DSGD-AC)*, which maintains controlled consensus diversity, we demonstrated that moderate and structured discrepancies among workers can act as implicit regularizer, leading to flatter minima and improved test performance.

Overall, the findings show our path from imitating centralized learning to challenging the conventional equivalence principle between decentralized and centralized learning. Rather than being a source of inefficiency, consensus diversity emerges as a tunable property that can improve robustness and generalization in decentralized training.

6 Design and implementation of Decent-DP

6.1 Motivation and design goal

Decentralized training, despite its theoretical advantages in communication efficiency and scalability, has historically faced significant implementation barriers. Existing frameworks, such as Bagua and BlueFog, often introduce heavy dependencies, complex installation procedures, and limited flexibility for experimental modifications. To facilitate reproducible research and practical deployment, we developed Decent-DP (stands for decentralized data parallelism), a PyTorch extension designed with three primary goals:

- **Efficiency:** The framework maximizes overlap between computation and communication, leverages heterogeneous communication links, and implements flexible communication topologies to reduce runtime.
- **Simplicity:** Minimal dependencies and a clean API ensure that researchers can integrate Decent-DP into existing PyTorch workflows with minimal code changes.
- **Flexibility:** Decent-DP exposes modular components (e.g., optimizer wrappers, topology managers) to enable experimentation with novel decentralized algorithms without rewriting core communication logic.

Decent-DP thus serves as both a research platform and a practical toolkit for decentralized deep learning on modern GPU clusters.

6.2 Implementation details

Synchronize before start Before the training, Decent-DP will perform a global broadcast to make sure all workers start from the exactly same copy of parameter.

Initialize the buckets Different from Pytorch Distributed Data Parallel [16], Decent-DP divides the parameters into buckets. Before the training, Decent-DP registers hooks (by `Tensor.register_post_accumulate_grad_hook`) on all parameters, which will be triggered during the backward pass when the corresponding gradient is computed. The hooks record the order of parameters used in the backward pass¹. After the orders are recorded, the extension will group the parameters into consecutive buckets and, in each bucket, it

- copies and re-maps the underlying storage to a continuous GPU memory for better efficiency when updating the parameters,
- creates the corresponding optimizer and learning rate scheduler,
- creates the corresponding buffer for asynchronous communication, and
- registers a hook on the last parameter in the bucket (`ddp_fn`).

Hook `ddp_fn` In the `ddp_fn` hook, since it is triggered when the gradient of the last parameter in the bucket is ready, it indicates the bucket is ready to update. In this hook, it

- receives the result from the communication operation launched asynchronously in the last iteration,
- copies the communication result to the parameter bucket (for the adaptive consensus introduced in Section 5.3.2, it simply replaces the copy with a weighted sum of local parameters and communication results),
- performs optimizer step and learning rate scheduler step,
- copies the updated parameter bucket to the communication buffer, and
- launch the communication operation asynchronously.

6.3 Usage and integration

The extension is open source on GitHub² and published on the Python Package Index (PyPI)³. The extension can be simply installed to the Python environment by the following command. The only dependencies of Decent-DP are `torch ≥ 2.1.0` and `loguru ≥ 0.7.3` for logging.

```
1 pip install decent-dp
```

Listing 6.1: Install Decent-DP

Here we demonstrates how the basic usage of Decent-DP compares with Pytorch Distributed Data Parallel.

¹In current implementation, it assumes the order is fixed and all the parameters are always used in all iterations

²<https://github.com/WangZesen/Decent-DP>

³<https://pypi.org/project/decent-dp/>

Change in model wrapper Listing 6.2 shows the how the model is wrapped in PyTorch DDP to make the model prepared for data parallelism.

```

1 import torch.distributed as dist
2 from torch.nn.parallel import DistributedDataParallel as DDP
3 from torchvision.models import resnet50
4
5 # initialize process group
6 dist.init_process_group()
7 # create the model (ResNet-50, for example)
8 model = resnet50()
9 # optimizer function takes the list of parameters, then
10 #   instantiates and returns an optimizer
11 # learning rate scheduler function takes an optimizer, then
12 #   instantiates and returns a LR scheduler
13 lr_scheduler = lr_scheduler_fn(optimizer)
14 # wrap the model with PyTorchDDP
15 model = DDP(
16     model,
17     gradient_as_bucket_view=True,
18     broadcast_buffers=True
19 )

```

Listing 6.2: Model wrapper in PyTorch DDP

Since Decent-DP handles the instantiation of optimizers and learning rate schedulers internally, Decent-DP takes the optimizer function and the LR scheduler function as input, and it additionally takes the name for the topology of decentralized communications. As shown in Listing 6.3, Decent-DP just needs minor changes for the wrapper.

```

1 import torch.distributed as dist
2 from decent_dp.ddp import DecentralizedDataParallel as DecentDP
3 from torchvision.models import resnet50
4
5 # initialize process group
6 dist.init_process_group()
7 # create the model
8 model = resnet50()
9 # wrap the model with Decentralized Data Parallel (Decent-DP)
10 model = DecentDP(
11     model,
12     optim_fn=optimizer_fn,
13     lr_scheduler_fn=lr_scheduler_fn,
14     topology="ring" # name of the selected topology
15 )

```

Listing 6.3: Model wrapper in Decent-DP

Change in training Listing 6.4 shows a typical example of training with PyTorch DDP.

```

1 for data, label in train_ds:
2     # forward pass
3     output = model(data)
4     # compute loss
5     loss = criterion(output, label)
6     # backward pass
7     loss.backward()
8     # optimizer update
9     optimizer.step()
10    # learning rate scheduler step
11    lr_scheduler.step()

```

Listing 6.4: Training example with PyTorch DDP

In Decent-DP, with the optimizer step and the learning scheduler step handled internally when executing the `ddp_fn`, the training loop becomes even more simple. Listing 6.5 showcases the training loop. By deleting `optimizer.step()` and `lr_scheduler.step()`, it becomes compatible with Decent-DP.

```

1 for data, label in train_ds:
2     # forward pass
3     output = model(data)
4     # compute loss
5     loss = criterion(output, label)
6     # backward pass with optimizer step and LR scheduler
7     # step handled internally in ddp_fn hook
8     loss.backward()

```

Listing 6.5: Training example with Decent-DP

Support for customized communication topologies Decent-DP supports customized communication topology. Listing 6.6 shows how the one-peer ring topology is defined using Decent-DP. The user needs to inherit and implement a `Topology` class which

- includes implementation for `_get_topo_edges`, which returns a list of lists of `Edge`, which defines the communication sub-groups in each iteration in the loop,
- is decorated with `@TopologyReg.register` which makes it visible when the name of the topology is passed to the wrapper DecentDP.

```

1 from decent_dp.topo import TopologyReg, Topology, Edge
2
3 @TopologyReg.register("ring")
4 class RingTopology(Topology):
5     def _get_topo_edges(self) → List[List[Edge]]:
6         if self._world_size % 2 ≠ 0:
7             logger.error("Ring topology is not supported "
8                          "for odd world size")
9             raise ValueError()

```

```

10
11     edges = [[], []]
12     # Odd iterations
13     for i in range(0, self._world_size, 2):
14         edges[0].append(
15             Edge(ranks=sorted([i, (i + 1) % self._world_size
16 ]), weight=0.5)
17         )
18     # Even iterations
19     for i in range(0, self._world_size, 2):
20         edges[1].append(
21             Edge(ranks=sorted([i, (i - 1 + self._world_size)
22 % self._world_size]), weight=0.5)
23         )
24     return edges

```

Listing 6.6: One-peer ring topology defined in Decent-DP

Support for adaptive consensus To support the adaptive consensus mechanism introduced in Section 5.3.2, Decent-DP exposes an interface to update the γ in Algorithm 3. The adaptive factor γ is set to 1 by default, which is equivalent to normal decentralized training.

```

1 model = DecentDP( ... )
2 for data, label in train_ds:
3     # ... training code
4
5     # update gamma
6     model.set_adaptive_factor(gamma)

```

Listing 6.7: Training example with adaptive consensus in Decent-DP

Easy start for implementations of decentralized algorithms The extension makes use of high-level APIs of PyTorch, and have everything implemented in Python, which allows easy manipulation to implement any decentralized algorithms that fit in the AWC computation-communication pattern.

6.4 Profiling results

To demonstrate the underlying mechanism of Decent-DP, we present the profiling results of training ResNet-50 on ImageNet-1K with PyTorch DDP and Decent-DP on two 4×A40 GPU nodes by using the profiler provided by PyTorch⁴.

As analyzed in Section 4, Figure 6.4.1 validates that there are always some idling time before the optimizer update because of the gradient communication. In the case of All-Reduce training, all communication operations take roughly same amount of time across iterations.

Figure 6.4.2 illustrates the characteristics of decentralized training:

- significant differences between the communication time for intra-node and inter-node communications can be observed,
- the communication time can be further hidden by the forward pass, and
- decentralized training facilitated by Decent-DP achieves better utilization of the computation resources of the GPU.

⁴https://docs.pytorch.org/tutorials/recipes/recipes/profiler_recipe.html

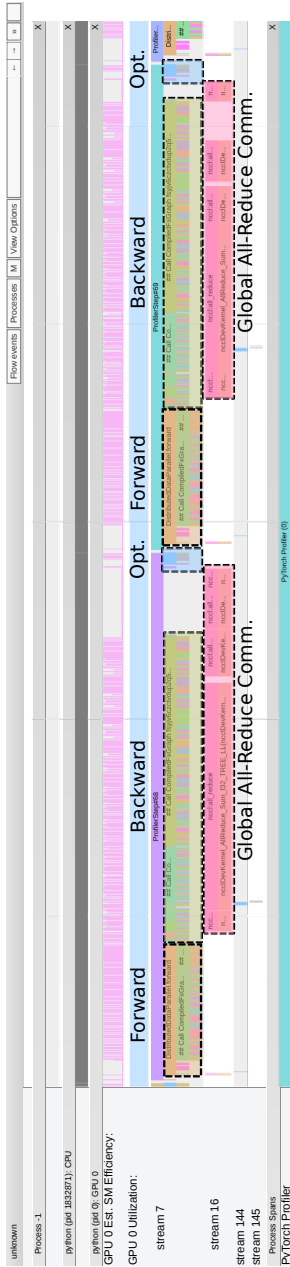


Figure 6.4.1: Profiling results for PyTorch DDP. Setup: $2 \times 4 \times A40$ GPU nodes interconnected by 25Gbps Ethernet.

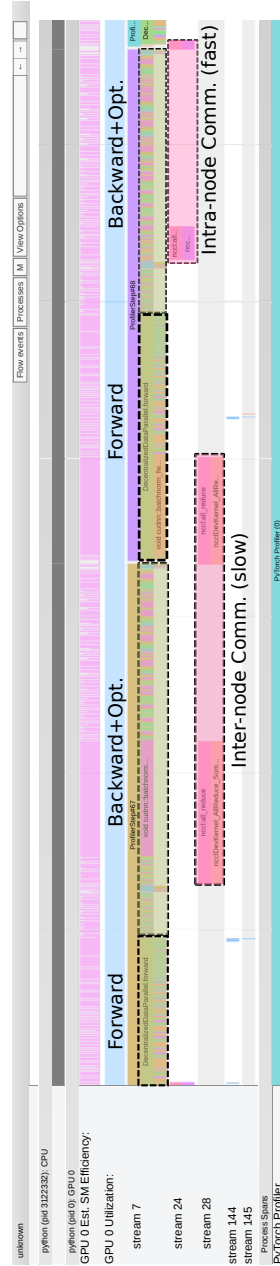


Figure 6.4.2: Profiling results for Decent-DP. Setup: $2 \times 4 \times A40$ GPU nodes interconnected by 25Gbps Ethernet. Topology: Alternating exponential ring.

6.5 Summary

This chapter introduced Decent-DP, a lightweight and modular PyTorch extension for decentralized deep learning. Designed for simplicity, efficiency, and modifiability, Decent-DP enables easy integration into existing workflows while supporting flexible communication topologies and adaptive consensus mechanisms. Its bucket-based parameter management and asynchronous communication design improve overlap between computation and communication, leading to reduced idle time and faster training compared to PyTorch DDP. Profiling results confirm Decent-DP's effectiveness as both a practical toolkit and a research platform for decentralized training.

7 Conclusions and future work

7.1 Concluding remarks

The widening gap between computational throughput and interconnect bandwidth makes data movement as one of the core bottlenecks in distributed deep learning. While centralized algorithms like All-Reduce have served as the bedrock of data parallelism, their reliance on global synchronization makes them increasingly inefficient and fragile as clusters scale to thousands of devices. This thesis has explored decentralized training as a viable alternative, lowering the three critical barriers that have historically prevented its widespread adoption: *ambiguity in efficiency, uncertainty in generalization, and complexity of implementation*.

7.2 Limitations and future research directions

While this licentiate thesis advances the state of decentralized learning, several avenues remain open for future exploration, particularly in the context of massive-scale Large Language Models (LLMs).

7.2.1 Decentralized Sharded Data Parallelism (FSDP)

This thesis focused primarily on the replicated data parallelism, where each worker holds a full copy of the model. However, frontier models now rely on Fully Sharded Data Parallelism (FSDP) [12] or ZeRO [13] to shard model states across devices. FSDP solves the memory constraint but exacerbates the communication bottleneck by requiring global All-Gather operations at every layer. A promising direction is to develop Decentralized FSDP, where model states are sharded across a sparse graph rather than the entire cluster. Research questions include:

- Can we reconstruct parameters for the forward pass using only partial information from neighbors?

- How does the "consensus error" manifest when parameters are sharded, and can it still be exploited for regularization?

7.2.2 Curvature-Aware decentralized optimization

A core finding of this thesis (Chapter 5) is that consensus error is not merely isotropic noise, but a structured perturbation aligned with the Hessian's high-curvature directions. This suggests that the communication topology itself acts as an implicit optimizer. Future research should move beyond analyzing this effect to *actively designing* algorithms that exploit it.

We envision a new class of curvature-aware decentralized optimizers that intentionally modulate the consensus mechanism to preserve or amplify favorable Hessian alignment. Potential directions include:

- **Anisotropic mixing:** Instead of using simple linear mixing using the doubly stochastic matrix $W^{(t)}$, the mixing matrix $W^{(t)}$ and the sent parameter $x_j^{(t)}$ could be dynamically adjusted based on local curvature estimates (excluding the update components on high-curvature directions from mixing). By relaxing the consensus constraint in sharp regions of the loss landscape, the optimizer could "inject" more Hessian-aligned noise to force the trajectory toward flatter minima.
- **Topological regularization:** Developing "mixing schedules" (analogous to learning rate schedules) where the graph connectivity evolves during training. A dense topology could be used in the initial phase for fast convergence, gradually transitioning to a sparse, high-noise topology in the final phase to encourage generalization and flatness.

7.2.3 Topology-aware hybrid parallelism

It is a natural future direction and also a need to verify the potential of decentralized training on applications with larger scale. By combining it with other parallelism (TP, PP, FSDP) and replacing the role of the data parallelism by Decent-DP, it is interesting to see how much speedup and improvement in utilization can be brought by decentralized training, and what could be the new challenges when integrating all these techniques as a comprehensive system.

Bibliography

- [1] Zesen Wang, Jiaojiao Zhang, Xuyang Wu, et al. “From promise to practice: realizing high-performance decentralized training”. In: *The Thirteenth International Conference on Learning Representations (ICLR 2025)*. 2025.
- [2] Zesen Wang and Mikael Johansson. “DSGD-AC: controlled consensus errors improve generalization in decentralized training”. *NeurIPS 2025 Workshop on Optimization for Machine Learning (OPT-2025)*. 2025.
- [3] Zesen Wang and Mikael Johansson. “Controlled disagreement improves generalization in decentralized training”. Under double-blind review. 2026.
- [4] Jacob Lindbäck, Zesen Wang, and Mikael Johansson. “Bringing regularized optimal transport to lightspeed: a splitting method adapted for GPUs”. In: *Advances in Neural Information Processing Systems* 36 (2023), pp. 26845–26871.
- [5] Aixin Liu, Bei Feng, Bing Xue, et al. “Deepseek-v3 technical report”. In: *arXiv preprint arXiv:2412.19437* (2024).
- [6] Bo Adler, Niket Agarwal, Ashwath Aithal, et al. “Nemotron-4 340b technical report”. In: *arXiv preprint arXiv:2406.11704* (2024).
- [7] Epoch AI. *Data on AI Models*. Accessed: 2025-10-23. July 2025. URL: <https://epoch.ai/data/ai-models>.
- [8] Priya Goyal, Piotr Dollár, Ross Girshick, et al. “Accurate, large minibatch sgd: Training imagenet in 1 hour”. In: *arXiv preprint arXiv:1706.02677* (2017).
- [9] Yanping Huang, Youlong Cheng, Ankur Bapna, et al. “Gpipe: Efficient training of giant neural networks using pipeline parallelism”. In: *Advances in neural information processing systems* 32 (2019).
- [10] Mohammad Shoeybi, Mostofa Patwary, Raul Puri, et al. “Megatron-lm: Training multi-billion parameter language models using model parallelism”. In: *arXiv preprint arXiv:1909.08053* (2019).
- [11] Deepak Narayanan, Mohammad Shoeybi, Jared Casper, et al. “Efficient large-scale language model training on gpu clusters using megatron-lm”. In:

BIBLIOGRAPHY

- Proceedings of the international conference for high performance computing, networking, storage and analysis*. 2021, pp. 1–15.
- [12] Yanli Zhao, Andrew Gu, Rohan Varma, et al. “Pytorch fsdp: experiences on scaling fully sharded data parallel”. In: *arXiv preprint arXiv:2304.11277* (2023).
- [13] Samyam Rajbhandari, Jeff Rasley, Olatunji Ruwase, et al. “Zero: Memory optimizations toward training trillion parameter models”. In: *SC20: International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE. 2020, pp. 1–16.
- [14] Rohan Anil, Andrew M Dai, Orhan Firat, et al. “Palm 2 technical report”. In: *arXiv preprint arXiv:2305.10403* (2023).
- [15] Jeffrey Dean, Greg Corrado, Rajat Monga, et al. “Large scale distributed deep networks”. In: *Advances in neural information processing systems* 25 (2012).
- [16] Shen Li, Yanli Zhao, Rohan Varma, et al. “PyTorch Distributed: Experiences on Accelerating Data Parallel Training”. In: *Proceedings of the VLDB Endowment* 13.12 (2020).
- [17] Epoch AI. *Data on Machine Learning Hardware*. Accessed: 2025-10-23. Oct. 2024. URL: <https://epoch.ai/data/machine-learning-hardware>.
- [18] Reza Olfati-Saber. “Distributed Kalman filtering for sensor networks”. In: *2007 46th IEEE conference on decision and control*. IEEE. 2007, pp. 5492–5498.
- [19] Bjorn Johansson, Maben Rabi, and Mikael Johansson. “A simple peer-to-peer algorithm for distributed optimization in sensor networks”. In: *2007 46th IEEE Conference on Decision and Control*. IEEE. 2007, pp. 4705–4710.
- [20] Angelia Nedic, Alex Olshevsky, Asuman Ozdaglar, et al. “On distributed averaging algorithms and quantization effects”. In: *IEEE Transactions on automatic control* 54.11 (2009), pp. 2506–2517.
- [21] Brendan McMahan, Eider Moore, Daniel Ramage, et al. “Communication-efficient learning of deep networks from decentralized data”. In: *Artificial intelligence and statistics*. PMLR. 2017, pp. 1273–1282.
- [22] Xiangru Lian, Ce Zhang, Huan Zhang, et al. “Can decentralized algorithms outperform centralized algorithms? a case study for decentralized parallel stochastic gradient descent”. In: *Advances in neural information processing systems* 30 (2017).
- [23] Mahmoud Assran, Nicolas Loizou, Nicolas Ballas, et al. “Stochastic gradient push for distributed deep learning”. In: *International Conference on Machine Learning*. PMLR. 2019, pp. 344–353.
- [24] Lingjing Kong, Tao Lin, Anastasia Koloskova, et al. “Consensus control for decentralized deep learning”. In: *International Conference on Machine Learning*. PMLR. 2021, pp. 5686–5696.
- [25] Kun Yuan, Yiming Chen, Xinmeng Huang, et al. “DecentLaM: Decentralized momentum SGD for large-batch deep training”. In: *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 2021, pp. 3029–3039.
- [26] Tongtian Zhu, Fengxiang He, Kaixuan Chen, et al. “Decentralized SGD and average-direction SAM are asymptotically equivalent”. In: *International Conference on Machine Learning*. PMLR. 2023, pp. 43005–43036.

- [27] Bicheng Ying, Kun Yuan, Hanbin Hu, et al. “Bluefog: Make decentralized algorithms practical for optimization and deep learning”. In: *arXiv preprint arXiv:2111.04287* (2021).
- [28] Shaoduo Gan, Jiawei Jiang, Binhang Yuan, et al. “Bagua: scaling up distributed learning with system relaxations”. In: *Proceedings of the VLDB Endowment* 15.4 (2021), pp. 804–813.
- [29] Alexander Sergeev and Mike Del Balso. “Horovod: fast and easy distributed deep learning in TensorFlow”. In: *arXiv preprint arXiv:1802.05799* (2018).
- [30] Yimin Jiang, Yibo Zhu, Chang Lan, et al. “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)*. 2020, pp. 463–479.
- [31] Ashish Vaswani, Noam Shazeer, Niki Parmar, et al. “Attention is all you need”. In: *Advances in neural information processing systems* 30 (2017).
- [32] Myle Ott, Sergey Edunov, Alexei Baevski, et al. “fairseq: A Fast, Extensible Toolkit for Sequence Modeling”. In: *Proceedings of NAACL-HLT 2019: Demonstrations*. 2019.
- [33] Niv Giladi, Shahar Gottlieb, Asaf Karnieli, et al. “DropCompute: simple and more robust distributed synchronous training via compute variance reduction”. In: *Advances in Neural Information Processing Systems* 36 (2024).
- [34] Nitish Shirish Keskar, Dheevatsa Mudigere, Jorge Nocedal, et al. “On Large-Batch Training for Deep Learning: Generalization Gap and Sharp Minima”. In: *International Conference on Learning Representations*. 2017.
- [35] Colin Raffel, Noam Shazeer, Adam Roberts, et al. “Exploring the limits of transfer learning with a unified text-to-text transformer”. In: *Journal of machine learning research* 21.140 (2020), pp. 1–67.
- [36] Hantian Ding, Zijian Wang, Giovanni Paolini, et al. “Fewer truncations improve language modeling”. In: *arXiv preprint arXiv:2404.10830* (2024).
- [37] Shaden Smith, Mostofa Patwary, Brandon Norick, et al. “Using deepspeed and megatron to train megatron-turing nlg 530b, a large-scale generative language model”. In: *arXiv preprint arXiv:2201.11990* (2022).
- [38] Jinkun Lin, Ziheng Jiang, Zuquan Song, et al. “Understanding Stragglers in Large Model Training Using What-if Analysis”. In: *arXiv preprint arXiv:2505.05713* (2025).
- [39] Tianyuan Wu, Lunxi Cao, Hanfeng Lu, et al. “Adaptra: Straggler-Resilient Hybrid-Parallel Training with Pipeline Adaptation”. In: *arXiv preprint arXiv:2504.19232* (2025).
- [40] Bicheng Ying, Kun Yuan, Yiming Chen, et al. “Exponential graph is provably efficient for decentralized deep training”. In: *Advances in Neural Information Processing Systems* 34 (2021), pp. 13975–13987.
- [41] Yuki Takezawa, Ryoma Sato, Han Bao, et al. “Beyond exponential graph: Communication-efficient topologies for decentralized learning via finite-time convergence”. In: *Advances in Neural Information Processing Systems* 36 (2023), pp. 76692–76717.
- [42] Kenta Niwa, Yuki Takezawa, Guoqiang Zhang, et al. “Revisiting 1-peer exponential graph for enhancing decentralized learning efficiency”. In: *The Thirty-ninth Annual Conference on Neural Information Processing Systems*. 2025.

BIBLIOGRAPHY

- [43] Binhang Yuan, Yongjun He, Jared Davis, et al. “Decentralized training of foundation models in heterogeneous environments”. In: *Advances in Neural Information Processing Systems* 35 (2022), pp. 25464–25477.
- [44] Wei Shi, Qing Ling, Gang Wu, et al. “Extra: An exact first-order algorithm for decentralized consensus optimization”. In: *SIAM Journal on Optimization* 25.2 (2015), pp. 944–966.
- [45] Angelia Nedic, Alex Olshevsky, and Wei Shi. “Achieving geometric convergence for distributed optimization over time-varying graphs”. In: *SIAM Journal on Optimization* 27.4 (2017), pp. 2597–2633.
- [46] Navjot Singh, Deepesh Data, Jemin George, et al. “SPARQ-SGD: Event-triggered and compressed communication in decentralized stochastic optimization”. In: *arXiv preprint arXiv:1910.14280* (2019).
- [47] Adel Nabli, Eugene Belilovsky, and Edouard Oyallon. “ $A^2C_iD^2$: Accelerating Asynchronous Communication in Decentralized Deep Learning”. In: *Advances in Neural Information Processing Systems* 36 (2023), pp. 47451–47474.
- [48] Jianyu Wang, Vinayak Tantia, Nicolas Ballas, et al. “Slowmo: Improving communication-efficient distributed SGD with slow momentum”. In: *arXiv preprint arXiv:1910.00643* (2019).
- [49] Arthur Douillard, Qixuan Feng, Andrei Alex Rusu, et al. “DiLoCo: Distributed Low-Communication Training of Language Models”. In: *2nd Workshop on Advancing Neural Network Training: Computational Efficiency, Scalability, and Resource Optimization (WANT@ ICML 2024)*.
- [50] Sami Jaghour, Jack Min Ong, and Johannes Hagemann. “Opendiloco: An open-source framework for globally distributed low-communication training”. In: *arXiv preprint arXiv:2407.07852* (2024).
- [51] Shi Pu and Angelia Nedić. “Distributed stochastic gradient tracking methods”. In: *Mathematical Programming* 187.1 (2021), pp. 409–457.
- [52] Yuki Takezawa, Han Bao, Kenta Niwa, et al. “Momentum tracking: Momentum acceleration for decentralized deep learning on heterogeneous data”. In: *arXiv preprint arXiv:2209.15505* (2022).
- [53] Tongtian Zhu, Fengxiang He, Lan Zhang, et al. “Topology-aware generalization of decentralized SGD”. In: *International Conference on Machine Learning*. PMLR. 2022, pp. 27479–27503.
- [54] Sulaiman A Alghunaim and Kun Yuan. “A unified and refined convergence analysis for non-convex decentralized learning”. In: *IEEE Transactions on Signal Processing* 70 (2022), pp. 3264–3279.
- [55] Daniel J Beutel, Taner Topal, Akhil Mathur, et al. “Flower: A friendly federated learning research framework”. In: *arXiv preprint arXiv:2007.14390* (2020).
- [56] Max Ryabinin and Anton Gusev. “Towards Crowdsourced Training of Large Neural Networks using Decentralized Mixture-of-Experts”. In: *Advances in Neural Information Processing Systems*. Vol. 33. 2020. URL: <https://proceedings.neurips.cc/paper/2020/file/25ddc0f8c9d3e22e03d3076f98d83cb2-Paper.pdf>.
- [57] Sepp Hochreiter and Jürgen Schmidhuber. “Flat minima”. In: *Neural computation* 9.1 (1997), pp. 1–42.
- [58] Laurent Dinh, Razvan Pascanu, Samy Bengio, et al. “Sharp minima can generalize for deep nets”. In: *International Conference on Machine Learning*. PMLR. 2017, pp. 1019–1028.

- [59] Pierre Foret, Ariel Kleiner, Hossein Mobahi, et al. “Sharpness-aware Minimization for Efficiently Improving Generalization”. In: *International Conference on Learning Representations*. 2021.
- [60] Jungmin Kwon, Jeongseop Kim, Hyunseo Park, et al. “Asam: Adaptive sharpness-aware minimization for scale-invariant learning of deep neural networks”. In: *International conference on machine learning*. PMLR. 2021, pp. 5905–5914.
- [61] Devansh Bisla, Jing Wang, and Anna Choromanska. “Low-pass filtering SGD for recovering flat optima in the deep learning optimization landscape”. In: *International Conference on Artificial Intelligence and Statistics*. PMLR. 2022, pp. 8299–8339.
- [62] Yong Liu, Siqi Mai, Xiangning Chen, et al. “Towards efficient and scalable sharpness-aware minimization”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2022, pp. 12360–12370.
- [63] Juntang Zhuang, Boqing Gong, Liangzhe Yuan, et al. “Surrogate Gap Minimization Improves Sharpness-Aware Training”. In: *International Conference on Learning Representations*. 2022.
- [64] Jiawei Du, Daquan Zhou, Jiashi Feng, et al. “Sharpness-aware training for free”. In: *Advances in Neural Information Processing Systems* 35 (2022), pp. 23439–23451.
- [65] Marlon Becker, Frederick Alrock, and Benjamin Risse. “Momentum-sam: Sharpness aware minimization without computational overhead”. In: *arXiv preprint arXiv:2401.12033* (2024).
- [66] Maksym Andriushchenko, Francesco Croce, Maximilian Müller, et al. “A Modern Look at the Relationship between Sharpness and Generalization”. In: *International Conference on Machine Learning*. PMLR. 2023, pp. 840–902.
- [67] Kaiyue Wen, Zhiyuan Li, and Tengyu Ma. “Sharpness minimization algorithms do not only minimize sharpness to achieve better generalization”. In: *Advances in Neural Information Processing Systems* 36 (2023), pp. 1024–1035.
- [68] Kishore Papineni, Salim Roukos, Todd Ward, et al. “Bleu: a method for automatic evaluation of machine translation”. In: *Proceedings of the 40th annual meeting of the Association for Computational Linguistics*. 2002, pp. 311–318.
- [69] Satanjeev Banerjee and Alon Lavie. “METEOR: An automatic metric for MT evaluation with improved correlation with human judgments”. In: *Proceedings of the acl workshop on intrinsic and extrinsic evaluation measures for machine translation and/or summarization*. 2005, pp. 65–72.
- [70] Lin Xiao and Stephen Boyd. “Fast linear iterations for distributed averaging”. In: *Systems & Control Letters* 53.1 (2004), pp. 65–78.
- [71] Guodong Shi, Bo Li, Mikael Johansson, et al. “Finite-time convergent gossiping”. In: *IEEE/ACM Transactions on Networking* 24.5 (2015), pp. 2782–2794.
- [72] Angelia Nedić, Alex Olshevsky, and Michael G Rabbat. “Network topology and computation tradeoffs in decentralized optimization”. In: *Proceedings of the IEEE* 106.5 (2018), pp. 953–976.
- [73] Yuki Takezawa, Ryoma Sato, Han Bao, et al. “Beyond exponential graph: Communication-efficient topologies for decentralized learning via finite-time

BIBLIOGRAPHY

- convergence”. In: *Advances in Neural Information Processing Systems 36* (2024).
- [74] Torsten Hoefler, Timo Schneider, and Andrew Lumsdaine. “Characterizing the influence of system noise on large-scale applications by simulation”. In: *SC’10: Proceedings of the 2010 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE, 2010, pp. 1–11.
- [75] Jared Kaplan, Sam McCandlish, Tom Henighan, et al. “Scaling laws for neural language models”. In: *arXiv preprint arXiv:2001.08361* (2020).
- [76] Ondřej Bojar, Christian Buck, Christian Federmann, et al. “Findings of the 2014 workshop on statistical machine translation”. In: *Proceedings of the ninth workshop on statistical machine translation*. 2014, pp. 12–58.
- [77] Kaiming He, Xiangyu Zhang, Shaoqing Ren, et al. “Deep residual learning for image recognition”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 770–778.
- [78] Jia Deng, Wei Dong, Richard Socher, et al. “Imagenet: A large-scale hierarchical image database”. In: *2009 IEEE conference on computer vision and pattern recognition*. Ieee, 2009, pp. 248–255.
- [79] Alec Radford, Jeffrey Wu, Rewon Child, et al. “Language models are unsupervised multitask learners”. In: *OpenAI blog 1.8* (2019), p. 9.
- [80] Aaron Gokaslan, Vanya Cohen, Ellie Pavlick, et al. *OpenWebText Corpus*. <http://Skylion007.github.io/OpenWebTextCorpus>. 2019.
- [81] Diederik Kinga, Jimmy Ba Adam, et al. “A method for stochastic optimization”. In: *International conference on learning representations (ICLR)*. Vol. 5. 6. California; 2015.
- [82] Alexandre Défossez, Léon Bottou, Francis Bach, et al. “A simple convergence proof of adam and adagrad”. In: *arXiv preprint arXiv:2003.02395* (2020).
- [83] Sergey Zagoruyko and Nikos Komodakis. “Wide residual networks”. In: *arXiv preprint arXiv:1605.07146* (2016).
- [84] Alex Krizhevsky, Geoffrey Hinton, et al. “Learning multiple layers of features from tiny images”. In: (2009).
- [85] Ilya Loshchilov and Frank Hutter. “SGDR: Stochastic gradient descent with warm restarts”. In: *arXiv preprint arXiv:1608.03983* (2016).
- [86] Akhilesh Gotmare, Nitish Shirish Keskar, Caiming Xiong, et al. “A closer look at deep learning heuristics: Learning rate restarts, warmup and distillation”. In: *arXiv preprint arXiv:1810.13243* (2018).
- [87] Dayal Singh Kalra and Maissam Barkeshli. “Why warmup the learning rate? underlying mechanisms and improvements”. In: *Advances in Neural Information Processing Systems 37* (2024), pp. 111760–111801.
- [88] Tao Li, Qinghua Tao, Weihao Yan, et al. “Revisiting random weight perturbation for efficiently improving generalization”. In: *arXiv preprint arXiv:2404.00357* (2024).
- [89] Guy Gur-Ari, Daniel A Roberts, and Ethan Dyer. “Gradient descent happens in a tiny subspace”. In: *arXiv preprint arXiv:1812.04754* (2018).
- [90] Minhak Song, Kwangjun Ahn, and Chulhee Yun. “Does SGD really happen in tiny subspaces?” In: *arXiv preprint arXiv:2405.16002* (2024).

- [91] Liu Ziyin, Kangqiao Liu, Takashi Mori, et al. “Strength of Minibatch Noise in SGD”. In: *International Conference on Learning Representations*. 2022. URL: <https://arxiv.org/abs/2102.05375>.
- [92] Lei Wu, Mingze Wang, and Weijie J. Su. “The Alignment Property of SGD Noise and How It Helps Select Flat Minima: A Stability Analysis”. In: *Advances in Neural Information Processing Systems*. Vol. 35. 2022, pp. 16843–16857. URL: <https://arxiv.org/abs/2207.02628>.
- [93] Takashi Mori, Liu Ziyin, Kangqiao Liu, et al. “Power-Law Escape Rate of SGD”. In: *Proceedings of the 39th International Conference on Machine Learning*. Vol. 162. Proceedings of Machine Learning Research. 2022, pp. 15959–15975. URL: <https://proceedings.mlr.press/v162/mori22a.html>.
- [94] Haocheng Luo, Tuan Truong, Tung Pham, et al. “Explicit eigenvalue regularization improves sharpness-aware minimization”. In: *Advances in Neural Information Processing Systems* 37 (2024), pp. 4424–4453.
- [95] Hongyang R Zhang, Dongyue Li, and Haotian Ju. “Noise stability optimization for finding flat minima: A Hessian-based regularization approach”. In: *arXiv preprint arXiv:2306.08553* (2023).