EFFICIENT DISTRIBUTED TRAINING WITH SUBNETWORKS

Zafir Khalid

A thesis
in
The Department
of
Computer Science and Software Engineering

Presented in Partial Fulfillment of the Requirements
For the Degree of Master of Science (Computer Science) at
Concordia University
Montréal, Québec, Canada

April 2026

Concordia University
School of Graduate Studies

This is to certify that the thesis prepared

By: Zafir Khalid
Entitled: **Efficient distributed training with subnetworks**

and submitted in partial fulfillment of the requirements for the degree of

Master of Science (Computer Science)

complies with the regulations of this University and meets the accepted standards with respect to originality and quality.

Signed by the final examining committee:

Dr. Yiming Xiao — Chair

Dr. Micro Ravanelli — Examiner

Dr. Eugene Belilovsky — Supervisor

Approved by

Dr. Leila Kosseim, Graduate Program Director
Department of Computer Science and Software Engineering

_____ 2026

Dr. Mourad Debbabi, Dean
Faculty of Engineering and Computer Science

Abstract

Efficient distributed training with subnetworks

Zafir Khalid

Pre-training large neural networks at scale places significant memory and communication demands on modern accelerators. As model sizes continue to grow faster than available device memory, efficiently distributing training across multiple hardware devices has become increasingly challenging. In practice, this challenge is shaped not only by theoretical scaling limits but also by system-level constraints such as network bandwidth, parameter redundancy, and computational overhead.

This thesis investigates a novel distributed training approach termed Model Parallelism with Subnetwork Data Parallelism, which introduces Subnetwork Data Parallelism (SDP) as a memory-efficient alternative to classical distributed data parallel (DDP) training. SDP partitions a model into structured, end-to-end subnetworks that are trained independently across workers without exchanging activations, thereby reducing per-device memory usage and communication costs.

The work examines two complementary masking regimes within the SDP framework. Backward masking applies sparsity exclusively during gradient computation, preserving unbiased gradient estimates and providing a theoretically grounded baseline, while forward masking extends sparsity to the forward pass, enabling further reductions in memory and computational cost while introducing additional regularization. In addition, two structured subnetwork construction strategies, neuron-level and block-level masking are explored across both convolutional neural networks and transformer-based architectures.

The proposed approach is evaluated through extensive experiments on image classification benchmarks, including CIFAR-10 and CIFAR-100, as well as large-scale language model pre-training on the FineWeb dataset. The results demonstrate that Subnetwork Data Parallelism achieves substantial memory savings while maintaining, and in some cases improving, performance relative to standard data-parallel training.

These findings highlight its practicality for training large models under constrained memory budgets.

Acknowledgments

I would like to take this moment to extend my deepest appreciation to my advisor, Dr. Eugene Belilovsky and my collaborator, Vaibhav Singh. Thank you for your guidance and mentorship throughout the course of this work.

This thesis marks the end of my time in academia. From preschool to here, between then and now - I owe in part to you.

My parents - without your constant support I wouldn't be half the person I am today. Thank you for your unbridled love that got me to where I am today.

My brothers - times when I couldn't see a way forward, you made one. Thank you for your unwavering encouragement and guidance that pushed me through even the toughest of times.

My friends - it was you who made this whole journey worthwhile. Thank you for keeping me grounded in times when I needed to be.

And lastly, my Aina.

When the rest of life is through, I will look back at this and think of how none of this would be possible without you. In ways you don't know and in ways I can't explain - you helped piece together the parts that got me here. Thank you for all that you've done and all that you'll do for the rest of life that's yet to come.

– Zafir

Contents

List of Figures

List of Tables

List of Abbreviations

CNN Convolutional Neural Network

DDP Distributed Data Parallel

FLOPs Floating Point Operations

FSDP Fully Sharded Data Parallel

LLM Large Language Model

MLP Multi-Layer Perceptron

NN Neural Network

ReLU Rectified Linear Unit

SDP Subnetwork Data Parallelism

SGD Stochastic Gradient Descent

Chapter 1

Introduction

1.1 Overview

Large-scale neural networks have become a central component of modern machine learning systems, driving progress across domains such as computer vision, natural language processing, speech recognition, and scientific computing. Models ranging from hundreds of millions or even up to trillions of parameters now routinely achieve state of the art performance on complex tasks (Team et al., 2026; Kaplan et al., 2020; Brown et al., 2020). However, this progress has come at the cost of rapidly increasing computational and memory requirements, placing significant strain on modern accelerator hardware. As a result, the ability to efficiently train large neural networks has become a critical challenge in both academic research and industrial practice.

To address these demands, distributed training techniques have become the standard approach for scaling neural network training across multiple hardware devices. Methods such as Distributed Data Parallelism (DDP) (Li et al., 2020) replicate models across workers and synchronize gradients during training, enabling faster convergence through increased batch sizes. While effective, these approaches incur substantial memory overhead due to full model replication and additional storage for gradients, optimizer states, and activations. As model sizes continue to grow faster than accelerator memory, such redundancy increasingly limits scalability, even in environments with abundant computational resources.

Beyond theoretical considerations, practical distributed training is further constrained by system-level factors including interconnect bandwidth, communication latency, and synchronization overhead. Techniques such as model parallelism and pipeline parallelism (Huang et al., 2019; Harlap et al., 2018) attempt to alleviate

memory bottlenecks by partitioning models across devices, but often require frequent activation exchanges and high-bandwidth interconnects to remain efficient. These requirements introduce additional complexity and can reduce training efficiency, particularly in settings where communication is a bottleneck or hardware resources are heterogeneous.

Recent research has therefore explored alternative strategies that trade full model replication for structured sparsity or partial model execution. Approaches based on subnetworks, masking, or selective parameter updates suggest that it may be possible to reduce memory usage and communication costs without significantly degrading training performance. However, many existing methods focus primarily on reducing communication or computation, rather than addressing the dominant memory costs associated with optimizer states and gradient storage during large-scale training.

This thesis builds upon these ideas by investigating a hybrid distributed training paradigm — Model Parallelism with Subnetwork Data Parallelism — that introduces Subnetwork Data Parallelism (SDP) as a memory-efficient alternative to classical data-parallel training. Rather than replicating the full model on each worker, SDP assigns structured, end-to-end subnetworks to different devices, allowing each worker to train only a subset of model parameters while avoiding activation exchange between devices. Through carefully designed masking strategies and parameter synchronization, the approach enables substantial reductions in per-device memory usage while preserving effective optimization behavior.

The work explores multiple instantiations of this framework, including forward and backward masking regimes, as well as neuron-level and block-level subnetwork construction strategies. These methods are evaluated across convolutional neural networks and transformer architectures, with experiments spanning image classification benchmarks and large-scale language model pre-training. By combining theoretical analysis with empirical evaluation, this thesis aims to demonstrate that Subnetwork Data Parallelism provides a practical and scalable pathway for training increasingly large models under constrained memory budgets.

1.2 Contributions

This thesis investigates Subnetwork Data Parallelism (SDP) as a memory-efficient paradigm and also introduces the idea of "Model Parallelism with Subnetwork Data Parallelism" for distributed training of deep neural networks. The work explores both algorithmic and practical aspects, including masking strategies,

subnetwork construction, theoretical properties, and empirical evaluation across vision and language modeling tasks. The primary goal is to attempt to reduce memory requirements while maintaining competitive optimization and generalization performance. The main contributions of this work are as follows:

- We provide a systematic investigation of a distributed training strategy that reduces memory usage by assigning structured subnetworks of a model to different devices. This approach enables large-scale training without requiring full model replication or inter-device activation exchange. More on this can be seen in chapter 4.
- In section 5.2 we show a comparative study of forward-masked and backward-masked training regimes, analyzing their impact on optimization dynamics, memory consumption, and overall training efficiency.
- We design and evaluate structured subnetwork construction strategies at both the neuron and block levels. Through extensive experimentation, we demonstrate that carefully structured subnetworks can maintain strong performance while significantly reducing per-device memory requirements. Work around this is covered in chapter 4.5.
- We provide extensive empirical validation across multiple vision tasks using ResNet and Swin Transformer architectures on CIFAR-10 and CIFAR-100 datasets. These experiments quantify the trade-offs between memory savings and model performance under different masking strategies. The experimentation is covered in chapter 5.
- We also evaluate SDP in the context of language modeling experiments, examining how subnetwork masking strategies affect training behavior and performance in transformer-based models. This work is covered in section 5.5.
- We analyze the theoretical properties of backward-masked SDP and characterize its convergence behavior, highlighting how mask structure and parameter overlap influence optimization quality. The theoretical analysis is covered in the Appendix.

1.2.1 Personal Contributions

During the course of my Master's program, I was mainly involved in the work done in this research project. In particular I was responsible for the development, empirical

evaluation of the vision components of our proposed framework, and the quantitative analysis.

- Implemented and evaluated many initial versions of the algorithm.
- The implementation and refinement of the codebase for the various masking strategies was carried out jointly with Vaibhav Singh.
- I conducted all experiments involving Swin Transformer architectures on the CIFAR-10 and CIFAR-100 datasets. Experiments with ResNet models were conducted together with Vaibhav Singh.
- I performed the bulk of the work around the quantitative analysis presented in Section 5.3.
- I wrote the experiments and results sections of the workshop paper that was later published at ES-FoMo at ICML 2025.

1.2.2 Collaborator Contributions

- Eugene Belilovsky provided the research roadmap and overall guidance that shaped the direction of the project.
- Vaibhav Singh individually conducted the language modeling experiments.
- Edouard Oyallon developed the theoretical analysis and mathematical proof characterizing the convergence behavior of backward-masked SDP. Which is given in the Appendix.

1.3 Thesis Outline

Chapter 2 presents the prerequisite concepts and background material required to understand the methodology developed in this thesis. Chapter 3 reviews existing research in distributed and large-scale deep learning, situating the proposed approach within the broader literature and highlighting its relation to prior work. Chapter 4 introduces the core methodology of Model Parallelism with Subnetwork Data Parallelism, detailing the proposed framework and its key components. Chapter 5 presents experimental results evaluating the proposed approach and examines its behavior across different architectures, datasets, and training configurations. Chapter

6 concludes the thesis with a discussion of the main findings, limitations, and directions for future research.

Chapter 2

Background

In this chapter, we explain briefly the background materials related to the works presented in this manuscript.

2.1 Distributed Training of Neural Networks

The continued growth in model size and dataset scale has made distributed training a fundamental component of modern deep learning systems. State-of-the-art neural networks in domains such as computer vision and natural language processing often contain hundreds of millions or billions of parameters, exceeding the memory and compute capacity of a single accelerator. Distributed training enables such models to be trained by coordinating computation across multiple workers, typically GPUs or accelerator nodes, allowing learning to scale beyond single-device limits.

At a high level, distributed training aims to minimize a global objective function by decomposing the optimization process across n workers. Let $\theta \in \mathbb{R}^d$ denote the model parameters and let $\mathcal{L}(\theta)$ represent the expected loss. Each worker independently samples mini-batches from the dataset and computes stochastic gradient estimates with respect to the current parameters. These local updates are then combined through a synchronization mechanism to form a global update. The design of this synchronization step plays a crucial role in determining both the efficiency and convergence behavior of the distributed algorithm.

While distributed training is often motivated by the desire for increased computational throughput, memory constraints are an equally important driving factor. Training time memory usage extends beyond model parameters to include intermediate activations, gradient buffers, and optimizer state. As architectures become deeper and training regimes involve longer sequences or larger batch sizes,

memory consumption can grow rapidly. In many practical settings, memory becomes the dominant bottleneck even when sufficient compute resources are available. Consequently, distributed training strategies must carefully manage how memory is allocated and replicated across workers.

In summary, distributed training provides the essential infrastructure for scaling deep neural networks to modern problem sizes. However, it introduces inherent trade-offs between memory usage, communication cost, and optimization efficiency. These trade-offs strongly influence the design of parallelism strategies for deep learning, which are discussed in the following section and form the basis for the memory efficient training approach developed in this thesis.

2.2 Parallelism Strategies for Deep Learning

Distributed training frameworks rely on different forms of parallelism to scale neural network optimization across multiple devices. These strategies differ in how model parameters, data, and computation are partitioned, and they exhibit distinct trade-offs in terms of memory usage, communication overhead, and implementation complexity. This section reviews the principal parallelism paradigms used in modern deep learning systems.

2.2.1 Distributed Data Parallelism

Distributed data parallelism (DDP) is the most widely adopted approach for distributed training. In this setting, each worker maintains a full replica of the model parameters $\theta \in \mathbb{R}^d$ and processes a different mini-batch of data. Let $\mathcal{B}_i$ denote the mini-batch sampled by worker i at a given iteration. Each worker computes a local stochastic gradient

$$g_i = \nabla_\theta \mathcal{L}(\theta; \mathcal{B}_i),$$

which is then aggregated across workers, typically via an all-reduce operation, to form the global gradient

$$\bar{g} = \frac{1}{n} \sum_{i=1}^{n} g_i.$$

The parameters are updated synchronously using $\bar{g}$.

DDP is attractive due to its conceptual simplicity and strong convergence properties (Goyal et al., 2018), as it closely approximates training with a larger

effective batch size. However, it requires each worker to store a complete copy of the model parameters, gradients, and optimizer state, resulting in substantial memory redundancy. Furthermore, frequent gradient synchronization can incur significant communication costs, particularly as model size grows.

2.2.2 Model Parallelism

Model parallelism addresses memory limitations by partitioning the model itself across workers. Instead of replicating the full parameter vector θ, different subsets of parameters are assigned to different devices. A common form of model parallelism is pipeline parallelism, where consecutive layers or blocks of the network are placed on different workers. During training, activations are passed forward through the pipeline, and gradients are propagated backward in reverse order.

While model parallelism reduces per-worker parameter memory, it introduces new challenges. Forward and backward passes require exchanging activations and gradients between workers, leading to communication overhead that scales with activation size rather than parameter count. Pipeline parallelism can also suffer from inefficiencies such as idle time, commonly referred to as pipeline bubbles, especially when the number of micro-batches is limited.

Another class of model parallelism techniques partitions individual layers across workers, often referred to as tensor parallelism. These methods split large matrix multiplications along specific dimensions and require fine-grained synchronization during both forward and backward passes. Although effective for very large models, they typically demand high-bandwidth interconnects to achieve good performance.

2.2.2.1 Pipeline Parallelism

Pipeline parallelism partitions the network into K stages:

$$f(x) = f_K \circ f_{K-1} \circ \ldots \circ f_1(x),$$

where stage k resides on worker k. Instead of processing a single batch sequentially across stages, the batch is divided into M micro-batches. At steady state, multiple micro-batches occupy different pipeline stages simultaneously. (Harlap et al., 2018; Huang et al., 2019; DeepSeek-AI et al., 2025)

If each stage requires time T, total iteration time becomes approximately

$$T_{\text{pipeline}} \approx (K + M - 1)\, T,$$

where the term $K-1$ reflects pipeline warm-up and drain phases. When M is small relative to K, devices remain idle during these phases, creating *pipeline bubbles*. Increasing M improves utilization but increases activation memory, since multiple micro-batches must be stored concurrently.

Pipeline parallelism reduces parameter memory per device to roughly N/K, but introduces activation communication: each forward pass requires transmitting intermediate activations of size proportional to hidden dimension and sequence length. In large transformers, activation tensors may exceed parameter size, especially for long contexts.

2.2.2.2 Tensor Parallelism

Tensor parallelism partitions individual layers across devices (Shoeybi et al., 2019; Shazeer et al., 2018). Consider a linear layer

$$y = Wx, \quad W \in \mathbb{R}^{d_{\text{out}} \times d_{\text{in}}}.$$

Column-wise partitioning splits W along d_{out}:

$$W = \begin{bmatrix} W_1 \\ W_2 \\ \vdots \\ W_n \end{bmatrix}.$$

Each worker computes a partial output $y_i = W_i x$, followed by an all-gather. Alternatively, row-wise partitioning requires partial reductions during the forward pass.

Tensor parallelism reduces parameter memory by a factor of n, but requires fine-grained synchronization during both forward and backward passes. Communication occurs at every layer, making high-bandwidth interconnects essential for efficiency.

2.2.3 Sharding and Fully Sharded Data Parallelism

Sharding-based approaches seek to reduce memory redundancy in data parallel training by distributing different components of the training state across workers. In Fully Sharded Data Parallel (FSDP), model parameters, gradients, and optimizer states are partitioned across workers, such that no single worker stores the full training state at once (Rajbhandari et al., 2020; Zhao et al., 2023). At a given step, each worker materializes only the parameter shards required for its local computation.

Formally, if $\theta = [\theta^{(1)}, \ldots, \theta^{(n)}]$ denotes a partition of the parameter vector across n workers, then worker i stores only $\theta^{(i)}$ at rest, gathering other shards on demand for computation and releasing them afterward. This design substantially reduces per-worker memory usage while preserving the semantics of data-parallel optimization.

Despite these benefits, sharding approaches still rely on frequent communication to assemble parameters and synchronize gradients. Moreover, the dynamic movement of parameters and optimizer state increases system complexity and can introduce additional latency if communication protocols are not overlapped with computation.

2.2.4 Summary

Each parallelism strategy offers a different balance between memory efficiency, communication cost, and implementation complexity. Data parallelism prioritizes simplicity but incurs high memory redundancy, while model parallelism reduces parameter storage at the expense of activation communication. Sharding-based methods such as FSDP mitigate memory overhead but retain significant communication requirements. These limitations motivate alternative approaches that rethink how model components are distributed and synchronized during training.

2.3 Memory Bottlenecks in Large-Scale Training

Memory consumption is a primary constraint in large-scale neural network training and often determines the maximum feasible model size, batch size, and sequence length. Unlike inference, where memory is dominated by model parameters, training requires additional storage for gradients, optimizer state, and intermediate activations. The relative importance of these components depends strongly on both architectural design (e.g., CNN vs Transformer) and training configuration.

2.3.1 Parameter, Gradient, and Optimizer Memory

Let $\theta \in \mathbb{R}^d$ denote the parameter vector. During training, multiple tensors associated with θ coexist in memory.

In mixed precision training, parameters are commonly stored in FP16 or bfloat16 (2 bytes per parameter) for forward and backward passes, while a master copy may be maintained in FP32 for numerical stability (Micikevicius et al., 2018). Gradients are typically accumulated in FP32 (4 bytes per parameter). For Adam, two additional FP32 buffers are stored for the first and second moments (Kingma and Ba, 2014).

Under this standard configuration, the per-parameter memory cost is approximately

$$2d \text{ (FP16 weights)} + 4d \text{ (FP32 gradients)} + 8d \text{ (Adam states)} = 14d \text{ bytes.}$$

If parameters are stored directly in FP32, the cost increases further. In standard data-parallel training, this entire footprint is replicated on every worker, making memory scale linearly with model size and independent of the number of workers.

For very large language models, this term typically dominates the memory footprint.

2.3.1.1 Activation Memory: CNNs versus Transformers

Activation memory behaves very differently across architectures.

Convolutional Neural Networks (CNNs). For a convolutional layer l, activations take the form

$$a^{(l)} \in \mathbb{R}^{B \times C_l \times H_l \times W_l},$$

where B is batch size, C_l channels, and H_l, W_l spatial resolution.

Activation memory scales as

$$\sum_{l=1}^{L} B \cdot C_l \cdot H_l \cdot W_l \cdot s,$$

where s is bytes per element.

In early CNN layers, spatial dimensions (H_l, W_l) are large. Even when C_l is moderate, the $H_l W_l$ term can dominate. Consequently, for high-resolution vision models, activation memory may exceed parameter memory, especially when batch size is large.

Thus, CNN training is often activation-bound in memory.

Transformer Architectures. For transformers, activations at layer l typically have shape

$$a^{(l)} \in \mathbb{R}^{B \times T \times d_{\text{model}}},$$

where T is sequence length and d_{model} hidden dimension.

Activation memory across layers scales as

$$\sum_{l=1}^{L} B \cdot T \cdot d_{\text{model}} \cdot s.$$

Unlike CNNs, there is no spatial $H \times W$ expansion. Instead, memory grows linearly in sequence length T.

Additionally, self-attention introduces temporary tensors of size

$$\mathbb{R}^{B \times h \times T \times T},$$

for h attention heads. These tensors scale quadratically in T, but are typically not stored persistently across layers (depending on implementation).

For LLM pre-training regimes, d_{model} and L are large, but batch size B is often constrained by memory. In practice, for billion-scale transformers, optimizer state and parameter storage dominate activation memory unless sequence length is very large.

Therefore:

- CNNs are often activation-dominated.
- Large transformers are typically parameter/optimizer-dominated.

This distinction is important when evaluating memory reduction strategies, as reducing parameters and optimizer state has greater impact for transformers, whereas reducing activations can be more critical for CNNs or long-context transformers.

Communication Buffers and Transient Memory

Distributed training introduces additional transient memory overhead. During gradient synchronization (e.g., all-reduce), communication buffers are typically allocated in FP32. These buffers may temporarily double gradient memory during synchronization. In sharded approaches, parameter all-gather operations similarly require temporary buffers proportional to shard size.

Although transient, these buffers must fit within device memory and therefore affect practical scalability.

Overall Scaling Behavior

Combining the above components, total per-device memory during training can be approximated as

$$\text{Memory} \approx \underbrace{\mathcal{O}(d)}_{\text{parameters + optimizer}} + \underbrace{\mathcal{O}\left(\sum_l B \cdot \text{activation size}_l\right)}_{\text{activations}}.$$

For CNNs with large spatial maps, the second term may dominate. For large transformers and LLMs, the first term typically dominates.

As models scale in depth and width, and as sequence lengths increase, these components quickly exhaust accelerator memory even under reduced precision arithmetic. This motivates training strategies that reduce parameter replication, limit optimizer and gradient storage, or avoid materializing inactive components during training.

Chapter 3

Related Work

The following section covers closely related concepts that are in the same domain as the main work presented in this thesis.

3.1 Pipeline Parallelism

Pipeline parallelism has been extensively studied as a mechanism for scaling deep neural network training beyond the memory capacity of a single accelerator. Prior work has demonstrated that partitioning models into sequential stages enables training of larger architectures by distributing parameters across devices. As discussed in section 2.2.2, this strategy fundamentally trades parameter replication for inter-device activation communication.

A significant body of work has focused on improving the efficiency of pipelined execution. Methods such as GPipe (Huang et al., 2019) introduce micro-batching to overlap computation across pipeline stages and reduce idle time. Other approaches extend this idea by combining pipeline parallelism with intra-layer partitioning, as seen in systems that shard large matrix operations within individual layers. These techniques have proven effective in practice, particularly for large transformer-based models.

Despite these advances, prior work consistently identifies communication of intermediate activations as a central bottleneck in pipeline-parallel training. Unlike gradient synchronization in data-parallel settings, which has been the subject of extensive optimization through compression and delayed updates, activation communication remains difficult to reduce without altering model semantics. This limitation becomes especially pronounced for long-sequence models, where activation sizes scale linearly with sequence length.

Another commonly reported challenge is pipeline inefficiency arising from stage imbalance and pipeline bubbles. Even with careful partitioning and scheduling, hardware utilization can degrade when pipeline stages differ in computational cost or when the number of micro-batches is insufficient. Several works, one such being DualPipe (DeepSeek-AI et al., 2025), attempt to mitigate these effects through improved scheduling or architectural modifications, but such solutions often increase system complexity and impose additional constraints on model design.

More recently, alternative formulations of parallel training have been proposed that seek to reduce reliance on activation exchange altogether. These approaches are motivated by the observation that, while pipeline parallelism alleviates memory pressure, its communication pattern can limit scalability in bandwidth-constrained environments. As a result, there is growing interest in distributed training strategies that preserve end-to-end model structure on each worker while avoiding activation-level communication.

3.2 Fully Sharded and Zero-Redundancy Data Parallelism

Fully sharded and zero-redundancy approaches have emerged as a prominent line of work aimed at alleviating the memory inefficiencies of standard data-parallel training. Building on the core idea of distributing model states across workers, these methods have enabled the training of increasingly large models within fixed hardware budgets and are now widely adopted in large-scale systems.

A number of practical frameworks have operationalized these ideas by aggressively sharding parameters, gradients, and optimizer states, materializing only the subsets required for local computation. Such systems demonstrate that substantial reductions in per-device memory usage can be achieved without modifying the underlying optimization objective, making them attractive for scaling existing training pipelines.

However, prior work also highlights several limitations inherent to fully sharded training. Because model states are partitioned across workers, parameters must be gathered before computation and redistributed afterward, resulting in frequent communication during both forward and backward passes. While overlapping communication with computation can partially mask this cost, synchronization overhead remains a dominant factor, particularly for large models or when operating under limited network bandwidth.

In addition to communication overhead, sharded approaches introduce increased system complexity. Managing dynamic parameter movement, coordinating collective operations, and ensuring efficient memory reuse require sophisticated runtime support. These factors can complicate deployment and tuning, especially when combined with other parallelism strategies such as pipeline or tensor parallelism.

As a result, while fully sharded and zero-redundancy methods significantly reduce memory redundancy, they do not eliminate the fundamental reliance on parameter-level synchronization. This observation has motivated complementary approaches that seek to lower memory usage while reducing the frequency or volume of inter-device communication required during training.

3.3 Ensemble Learning and Related Parallel Training Methods

Ensemble learning constitutes a broad class of methods that improve model performance by training multiple models in parallel and combining their predictions or parameters. By encouraging diversity among individual learners, ensemble-based approaches often achieve better generalization and robustness than single-model training. In recent work, ensemble ideas have also been explored in distributed optimization settings, where parallelism is leveraged not only for scalability but also as a source of regularization.

Several studies investigate training multiple related models concurrently with periodic coordination mechanisms such as parameter averaging or alignment. These methods allow individual replicas to follow distinct optimization trajectories while maintaining a degree of coherence across the population. Such approaches have been shown to be effective in large-scale training regimes, where communication-efficient coordination is critical to overall system performance.

From a conceptual standpoint, training subnetworks on different workers bears resemblance to ensemble learning. Each worker optimizes a structurally valid but reduced model, resulting in a collection of correlated learners that jointly influence the optimization process. In particular, forward subnetwork masking can be interpreted as training an ensemble of overlapping models whose parameters are periodically synchronized. This perspective provides useful intuition for understanding the regularization effects observed in subnetwork-based training.

Despite these similarities, traditional ensemble methods are typically not designed

with memory efficiency as a primary objective. Most ensemble-based systems rely on maintaining multiple full model replicas, which can be prohibitively expensive when training large neural networks. In contrast, subnetwork-based approaches aim to reduce per-worker memory and computation by explicitly restricting the set of parameters and activations materialized on each device.

Overall, ensemble learning and related parallel training methods highlight the benefits of training multiple interacting models in parallel. While their primary goal differs from memory-efficient distributed training, the connections between ensemble behavior and subnetwork optimization offer valuable insight into how reduced models can be coordinated effectively at scale.

3.4 SWARM and Hybrid Parallelism Approaches

Hybrid parallelism strategies aim to combine the strengths of data parallelism and model parallelism in order to improve scalability and resource utilization. Among these, SWARM-style approaches have been proposed to address limitations of pipeline parallelism by altering how computation and communication are organized across devices.

SWARM parallelism (Ryabinin et al., 2023) assigns multiple devices to each pipeline stage and dynamically routes samples through the network. By replicating stages across workers, these methods seek to reduce idle time and mitigate pipeline bubbles while retaining the ability to scale to large models. This design allows SWARM to exploit both intra-stage data parallelism and inter-stage pipelining, offering improved throughput compared to conventional pipeline-parallel execution.

Despite these advantages, SWARM-based methods retain a key characteristic of pipeline parallelism: the need to exchange activations between stages. Although routing and replication strategies can reduce inefficiencies, intermediate activations must still be communicated across devices, which can become a performance bottleneck in bandwidth-constrained environments. As with traditional pipeline parallelism, the cost of activation communication grows with sequence length and feature dimensionality.

In addition, hybrid approaches such as SWARM introduce increased system complexity. Coordinating routing decisions, managing replicated stages, and balancing workloads across devices require sophisticated runtime support. These design choices can complicate deployment and tuning, particularly when scaling to large numbers of workers or heterogeneous hardware.

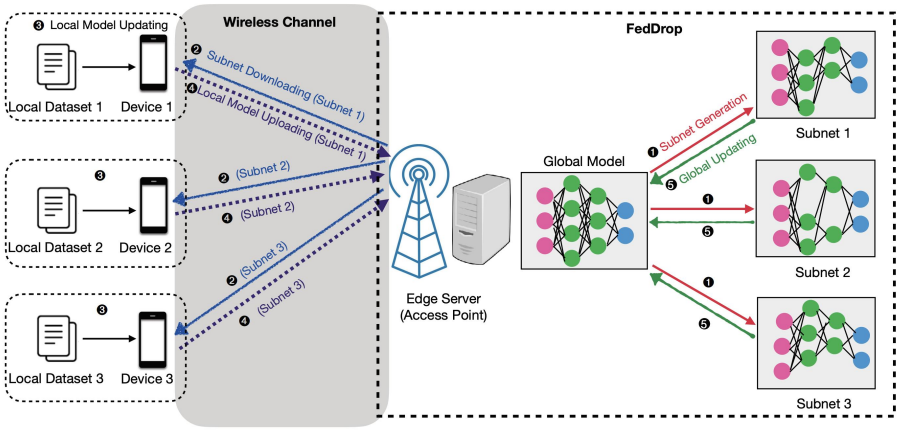


Figure 1: A schematic of FedDrop from (Wen et al., 2022)

Overall, SWARM and related hybrid parallelism methods represent an important step toward improving the efficiency of pipeline-based training. However, their continued reliance on activation-level communication highlights the broader challenge of scaling large models under memory and bandwidth constraints, motivating alternative distributed training strategies that minimize inter-device communication while preserving end-to-end model structure.

3.5 Federated Learning and Dropout-Based Subnetwork Training

Federated learning provides a distributed optimization framework in which training is performed across decentralized data sources, typically under constraints such as privacy, limited connectivity, and statistically heterogeneous (non-iid) data distributions (Konečný et al., 2016). Within this setting, a recurring theme is to reduce the per-client burden both in communication and in local computation to enable participation from resource constrained devices. Motivated by these constraints, multiple works investigate training only a *subnetwork* of a larger global model on each device (Caldas et al., 2018; Horvath et al., 2021; Guliani et al., 2022; Wen et al., 2022; Alam et al., 2022). In these approaches, subnetwork assignment (often realized through dropout-style masking) reduces the number of active parameters on each client, which can lower on-device compute and shrink the transmitted update size.

Importantly, the primary objective in this line of work is typically *communication*

and *device-compute* efficiency in heterogeneous and privacy-constrained environments, rather than minimizing memory usage for large-model training on homogeneous accelerators. Communication load in federated optimization has been studied extensively and can be mitigated by multi-step local training and communication-efficient update schemes, including compression and low-communication optimization variants (Reddi et al., 2021; Douillard et al., 2023; Wang et al., 2023). Consequently, federated subnetwork methods are often designed around communication bottlenecks and client limitations, whereas our focus is on reducing memory requirements that arise when training large models on memory-limited GPUs.

A closely related subset of federated subnetwork methods explicitly addresses *device heterogeneity* by varying model capacity across clients. FedRolex (Alam et al., 2022) and HeteroFL (Diao et al., 2021) assign models of different sizes to different clients, commonly via channel-level dropout or width-based scaling in convolutional networks, so that smaller devices train thinner subnetworks while more capable devices train larger ones. This design is well suited to federated scenarios where clients differ widely in compute and memory, but it is grounded in assumptions that differ from centralized distributed training. In particular, these methods operate under privacy and partial-participation constraints and often assume clients have access only to small local datasets, which introduces additional concerns such as non-iid effects and overfitting. In contrast, our setting assumes homogeneous workers and access to the full dataset per worker, which avoids client heterogeneity and local data scarcity effects and allows the training objective to emphasize per-node memory reduction.

Another practical distinction is that many federated subnetwork approaches employ *dynamic* subnetwork assignments over training. While dynamic masking can improve fairness or utilization across heterogeneous clients, it introduces additional coordination and communication overhead, which can be undesirable in non-federated environments where wall-clock time and systems simplicity are critical. Yuan et al. (2019) studied training with dynamic, non-overlapping subnetworks combined with local SGD, illustrating that subnetwork scheduling policies can strongly influence optimization behavior. However, such dynamic schemes can require additional synchronization to maintain consistency across workers.

Beyond federated learning, related work also explores selectively skipping gradient computation for certain components of a network to accelerate training. For example, Fagnou et al. (2025) examine skipping backward computation along residual paths, but this line of work does not directly address distributed settings nor does it

target memory reduction under FLOP-matched comparisons. In comparison, our approach emphasizes fixed, structured subnetwork assignments and considers both neuron-level and block-level masking, including in transformer-based architectures and large language model training. Moreover, unlike the above federated and dynamic-subnetwork approaches, our method includes a backward-masking regime, and to the best of our knowledge prior work in these federated subnetwork lines does not consider masking only the backward pass.

Overall, federated learning and dropout-based subnetwork training provide important insights into how reduced models can be assigned and aggregated in distributed systems (Konečný et al., 2016; Caldas et al., 2019; Horvath et al., 2021; Guliani et al., 2022; Wen et al., 2022; Alam et al., 2022). Nonetheless, their goals and assumptions, privacy constraints, non-iid data, client heterogeneity, and communication efficiency differ from the homogeneous, centralized training regime considered in this thesis. These differences motivate alternative approaches that target per-worker memory reduction while limiting coordination overhead and preserving scalable training dynamics.

Chapter 4

Model Parallelism with Subnetwork Data Parallelism

This chapter presents Model Parallelism with Subnetwork Data Parallelism, the central methodology developed in this thesis.

4.1 From Replicated Training to Subnetwork Model Parallelism

Section 2.3 established that, in conventional data-parallel training, the dominant memory cost arises from full parameter replication and associated optimizer state on every worker. While memory capacity is the primary bottleneck, large-scale distributed training also faces a second constraint: inter-device communication bandwidth.

In standard Distributed Data Parallel (DDP), each of the n workers maintains an identical copy of parameters $\theta \in \mathbb{R}^d$. After computing local gradients $g_i = \nabla_\theta \mathcal{L}_i(\theta)$, workers synchronize via an all-reduce operation,

$$g \;=\; \frac{1}{n} \sum_{i=1}^{n} g_i,$$

which requires exchanging d-dimensional tensors across devices at every optimization step. Under ring-based all-reduce, this corresponds to communication proportional to $2d$ scalars per worker per iteration (Patarasuk and Yuan, 2009; Tang et al., 2023). As d grows into the billions, this synchronization becomes a dominant cost even when memory is sufficient.

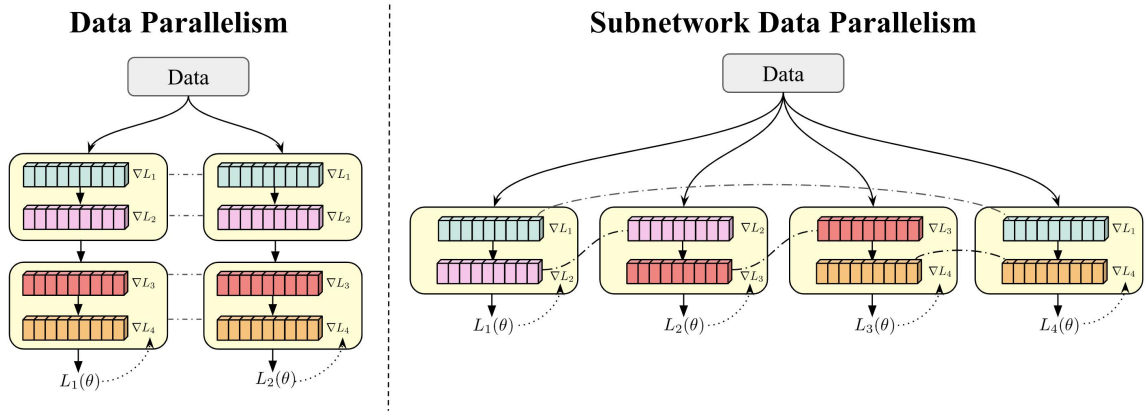


Figure 2: **Data Parallelism (DDP) vs. Subnetwork Data Parallelism (SDP).** *Left:* In data parallelism each GPU hosts a full replica, computes all layer gradients $\{\nabla L_1, \nabla L_2, \nabla L_3, \nabla L_4\}$, and all-reduces *all* parameters each step; per-GPU memory is approximately the full model (parameters + gradients + optimizer state + activations). *Right:* In SDP each GPU trains an end-to-end *subnetwork* (a subset of layers/neurons) with a local loss $L_k(\theta)$; only gradients of *shared* parameters are synchronized via masked averaging (dashed arcs). For a **parameter density per GPU** $C \in (0,1]$, both memory and communication per GPU scale as $\approx C \times$ DP, with no cross-GPU activation exchange. This enables fitting larger models or longer sequences under the same hardware budget and improves scalability when bandwidth or memory are bottlenecks; when $C = 1$ (all parameters on all GPUs), SDP reduces to standard DDP.

Model-parallel techniques address replication by partitioning θ across workers. Let $\theta = (\theta^{(1)}, \ldots, \theta^{(n)})$ denote a disjoint partition, where worker i stores only $\theta^{(i)}$. While this removes parameter replication, it introduces a different scaling bottleneck: activations must be exchanged between workers during the forward and backward passes. If $a^{(\ell)}$ denotes the activation at layer ℓ, then model-parallel execution requires transmitting tensors of size proportional to $|a^{(\ell)}|$ across devices at layer boundaries. For transformer architectures with large hidden dimension h and sequence length T, these activation tensors can be substantially larger than individual parameter shards, leading to bandwidth-bound training.

This chapter introduces *Model Parallelism with Subnetwork Data Parallelism* (SDP), which occupies a distinct point in the design space. Instead of replicating the full model (as in DDP) or partitioning layers with activation exchange (as in tensor or pipeline parallelism), each worker trains an *end-to-end subnetwork* - a structured subset of the global parameters. Let $m_i \in \{0,1\}^d$ denote a binary mask defining the parameters active on worker i. Worker i trains the masked model

$$\theta_i \;=\; m_i \odot \theta,$$

where $\odot$ denotes elementwise multiplication. Crucially, the forward and backward passes are executed locally without exchanging activations across devices.

Communication occurs only at synchronization points, where overlapping parameters across workers are averaged. Denoting $c_j = \sum_{i=1}^{n} m_{i,j}$ as the number of workers containing parameter coordinate j, the aggregated parameter update takes the form

$$\theta_j \leftarrow \begin{cases} \frac{1}{c_j} \sum_{i:m_{i,j}=1} \theta_{i,j}, & c_j > 0, \\ \theta_j, & c_j = 0. \end{cases}$$

Thus, unlike classical model parallelism, SDP avoids cross-device activation communication, and unlike DDP, it avoids full parameter replication.

This perspective can be interpreted as a structured, overlapping form of model parallelism: workers store different but partially intersecting parameter subsets, yet each subnetwork remains functionally complete from input to loss. The degree of overlap controls both memory usage and communication volume. If the average parameter density per worker is $C \in (0, 1]$, then both memory footprint and gradient communication scale proportionally to Cd, rather than d.

The remainder of this chapter formalizes this framework, introduces structured masking strategies, derives convergence guarantees under backward masking, and analyzes the resulting memory and communication trade-offs.

4.2 Subnetwork Data Parallelism: Core Idea and Training Procedure

Section 4.1 positioned Subnetwork Data Parallelism (SDP) as an alternative to both fully replicated data parallelism and activation-communicating model parallelism. We now describe the core idea and the resulting training loop at a conceptual level.

4.2.1 Core Idea

In standard data parallel training, each of the n workers maintains a complete copy of the model parameters $\theta \in \mathbb{R}^d$. In contrast, SDP assigns each worker a structured *subnetwork* - a subset of parameters that defines a functionally complete path from input to loss.

Let $\theta^{(t)}$ denote the global parameters at iteration t. Each worker i trains a masked version of the model,

$$\theta_i^{(t)} = m_i \odot \theta^{(t)},$$

where $m_i \in \{0,1\}^d$ specifies which parameters are active on that worker.

Crucially:

- Each worker executes a full forward and backward pass locally.
- No intermediate activations are exchanged across devices.
- Only overlapping parameters (or gradients) are synchronized.

Thus, every worker trains an end-to-end model, but different workers operate on partially overlapping subnetworks. The overlap ensures that information propagates globally through parameter averaging.

4.2.2 Training Loop

At each iteration t, the distributed training procedure consists of four steps:

1. Broadcast. Workers start from a common parameter vector $\theta^{(t)}$. In practice, this is already synchronized from the previous step.

2. Local Forward and Backward Pass. Worker i samples a minibatch $\xi_i^{(t)}$ and computes the stochastic gradient of its masked model:

$$g_i^{(t)} = \nabla_{\theta_i} \ell(\theta_i^{(t)}; \xi_i^{(t)}).$$

Only parameters active under m_i are materialized and updated.

3. Local Parameter Update. Each worker performs a local optimizer step (e.g., SGD or Adam) restricted to its active coordinates:

$$\theta_i^{(t+\frac{1}{2})} = \theta^{(t)} - \eta\, g_i^{(t)}.$$

4. Masked Synchronization. For parameters that appear on multiple workers, values are averaged:

$$\theta_j^{(t+1)} = \frac{1}{c_j} \sum_{i : m_{i,j}=1} \theta_{i,j}^{(t+\frac{1}{2})},$$

where c_j denotes the number of workers containing coordinate j.

Parameters present on only one worker remain unchanged by averaging.

4.2.3 Interpretation

This procedure can be interpreted as training a collection of overlapping subnetworks whose parameters are periodically aligned. Unlike classical model parallelism, no layer-wise activation communication is required. Unlike standard data parallelism, parameters and optimizer state are not fully replicated.

If the average parameter density per worker is $C \in (0, 1]$, then:

- Memory footprint scales as $\mathcal{O}(Cd)$ per worker.
- Communication volume per iteration scales as $\mathcal{O}(Cd)$.

The choice of masking strategy - how subnetworks are constructed and how overlap is controlled, determines the trade-off between efficiency, convergence behavior, and representational capacity. These design choices are formalized in the following section.

4.3 Generic Masking Framework

We now formalize Subnetwork Data Parallelism (SDP) using a masking abstraction.

4.3.1 Masking Matrix

Let $\theta \in \mathbb{R}^d$ denote the global parameter vector and let n be the number of workers. SDP is defined through a binary masking matrix

$$m' \in \{0, 1\}^{n \times d},$$

where $m_{i,j} = 1$ indicates that worker i contains parameter coordinate j.

The mask for worker i is the row vector

$$m'_i \in \{0, 1\}^d,$$

and the number of workers containing coordinate j is given by the column load

$$c_j = \sum_{i=1}^{n} m'_{i,j}.$$

Each worker stores and updates only its masked parameters:

$$\theta_i = m'_i \odot \theta,$$

where $\odot$ denotes elementwise multiplication.

The masking matrix is constructed such that each worker is assigned a fixed number of components. In particular, for a given per-worker budget p, the mask satisfies

$$\sum_{j=1}^{d} m'_{i,j} = p,$$

so that every worker trains a subnetwork of identical size.

Let m denote the total number of parameter coordinates. We define the parameter density

$$C = \frac{p}{m},$$

which represents the fraction of the global parameters assigned to each worker. When $C = 1$, the masking matrix reduces to the all-ones matrix and the method recovers standard data-parallel training. As C decreases, each worker stores and updates a smaller subset of parameters, reducing memory usage and synchronization volume proportionally.

4.3.2 Forward and Backward Masking

The framework distinguishes between forward masking and backward masking. Let

$$m^{\text{fwd}},\ m^{\text{bwd}} \in \{0,1\}^{n \times d}$$

denote the masks used during forward/backward computation and during gradient aggregation, respectively.

Forward-masked SDP. In this setting,

$$m^{\text{fwd}} = m^{\text{bwd}} = m.$$

Workers compute and update only the parameters specified by their mask.

Backward-masked SDP. Here,

$$m^{\text{fwd}} = \mathbf{1}, \qquad m^{\text{bwd}} = m,$$

where $\mathbf{1}$ denotes the all-ones mask. The full model participates in the forward pass, but only masked parameters are updated and synchronized.

4.3.3 Masked Aggregation

After local optimization steps, parameters are synchronized only across workers that share them. The aggregation operator is defined coordinatewise as

$$\theta_j \leftarrow \frac{1}{c_j} \sum_{i : m^{\text{bwd}}_{i,j} = 1} \theta_{i,j}.$$

Coordinates assigned to multiple workers are averaged; coordinates assigned to a single worker remain unchanged.

When m is the all-ones matrix, this reduces to standard data-parallel averaging. When masks are disjoint (i.e., $c_j = 1$ for all j), synchronization has no effect and training becomes fully partitioned.

4.3.4 Communication Scaling

The total number of parameter coordinates participating in synchronization per iteration is

$$\sum_{j=1}^{d} c_j = np = nCd.$$

Thus, per worker, communication scales proportionally to Cd rather than d, recovering classical data parallelism when $C = 1$ and reducing communication linearly with sparsity.

This masking abstraction provides the foundation for the structured subnetwork constructions introduced in the following section.

4.4 Subnetwork Data Parallelism with Structured Mask Construction

We instantiate Subnetwork Data Parallelism (SDP) by employing structured masks, which remove entire parameter groups including parameters, gradients, accumulators, and activations from each worker. We consider two strategies for instantiating subnetworks: Neuron-Level SDP (**N-SDP**) and Block-Level SDP (**B-SDP**), and also the backward-masked block variant $\mathbf{B}_b$**-SDP**.

4.4.1 Neuron Level SDP

Neuron-Level SDP (**N-SDP**) instantiates subnetworks by selectively removing neurons in fully connected layers (or channels in convolutional layers). Consider two successive layers (W^l, W^{l+1}) with

$$W^l : \mathbb{R}^{d_{l-1}} \to \mathbb{R}^{d_l}.$$

Dropping outputs of layer l naturally removes the corresponding inputs of layer $l{+}1$. For simplicity, we restrict to forward masking, where the same mask is applied in both directions:

$$\mathbf{m}_{\text{fwd}} = \mathbf{m}_{\text{bwd}} = \mathbf{m}.$$

Applying a mask $\mathbf{m}^l$ to layer l induces a consistent mask $\mathbf{m}^{l+1}$ on layer $l{+}1$. As a result,

$$(\mathbf{m}^l \odot W^l,\, W^{l+1}) \quad \text{and} \quad (\mathbf{m}^l \odot W^l,\, \mathbf{m}^{l+1} \odot W^{l+1})$$

produce identical outputs.

For example, if $W^l, W^{l+1} \in \mathbb{R}^{d\times d}$ and we mask a subset $J_{\text{mask}} \subset \{1, \dots, d\}$ of output neurons, then setting

$$\begin{aligned} m^l_{jk} &= 0, \qquad j \in J_{\text{mask}},\ k \in \{1, \dots, d\}, \\ m^{l+1}_{kj} &= 0, \qquad j \in J_{\text{mask}},\ k \in \{1, \dots, d\}, \end{aligned} \tag{1}$$

ensures that both layers remain consistent under the masking operation.

4.4.2 Block Level SDP

Block-Level SDP (**B-SDP**) forms subnetworks by removing entire blocks, particularly in architectures with skip connections. Let the model have L blocks $\{B^1, \dots, B^L\}$

with parameters $\boldsymbol{\theta}^{(l)}$. Each block has a binary mask $m^{(l)} \in \{0, 1\}$ denoting whether it is active. When $m^{(l)} = 0$, the block is skipped and its parameters excluded.

In residual architectures, this reduces to the identity mapping via the skip path, ensuring valid representations even when blocks are dropped. Formally, for a residual connection of the form

$$B^{(l)}(\mathbf{x}) + \mathbf{x},$$

the masked computation at block l is

$$\hat{B}^{(l)}(\mathbf{x}) = m^{(l)} B^{(l)}(\mathbf{x}) + \mathbf{x}. \tag{2}$$

4.4.3 Block Backwards SDP

We also consider the more general case of backward masking, where

$$\mathbf{m}_{\text{fwd}} = \mathbf{m}^{\text{uni}} \qquad \text{and} \qquad \mathbf{m}_{\text{bwd}} = \mathbf{m}.$$

In this instantiation, the block may be active during the forward pass but omitted during the backward pass. We refer to this block-level backward-masked setting as **$\mathbf{B}_b$-SDP**.

4.4.4 Memory, compute, and communication cost

Let N be the total parameter count and $C \in (0, 1]$ the per-worker density (fraction of coordinates selected by the mask). Consider bf16 parameters (2 bytes), fp32 gradients (4 bytes), and Adam accumulators in fp32 (8 bytes). Standard data parallel training requires approximately $14N$ bytes per worker (Rajbhandari et al., 2020; Micikevicius et al., 2018).

In forward masking, only the CN coordinates materialize parameters, gradients, and accumulators, using approximately $14CN$ bytes; for structured masks (channel/block level), activations and compute also scale approximately with C.

In backward masking, the full forward is computed, but gradients and accumulators are stored only for the CN active coordinates, giving

$$(2 + 12C)N$$

bytes. Activations scale approximately with C.

Communication cost is also reduced under SDP. In ring all-reduce, each worker

with N parameters sends and receives about $2N$ scalars per step, whereas SDP synchronizes only the CN active coordinates, reducing the cost to approximately $2CN$ (Patarasuk and Yuan, 2009; Tang et al., 2023). When masks differ across workers, each parameter block is reduced only within its subset of workers; this holds for both forward and backward masking.

4.5 Balanced Mask Construction for SDP

Let n be the number of workers, m the number of components (can be either a layer or a neuron), and $p \in \{0, \dots, m\}$ the per-worker budget. We construct a binary mask

$$\mathbf{M} \in \{0,1\}^{n \times m}, \qquad \mathbf{M}_{i,j} = 1 \;\Leftrightarrow\; \text{worker } i \text{ is assigned component } j.$$

Define the column loads $c_j = \sum_{i=1}^{n} M_{i,j}$ and the target load

$$c^\star = \left\lceil \frac{np}{m} \right\rceil.$$

The mask is constructed by a (probabilistic) procedure that enforces the exact per-worker budget

$$\sum_{j=1}^{m} M_{i,j} = p \qquad \text{for all } i,$$

and heuristically balances column loads by biasing assignments toward under-covered components, with the goal of approximately minimizing the imbalance

$$\min \max_{j \in \{1,\dots,m\}} \left| c_j - \frac{np}{m} \right|.$$

Algorithm 1 describes the structured mask construction.

Algorithm 1: Structured Mask Construction

Input: n, m, p
Output: $M \in \{0,1\}^{n \times m}$
Initialize $M \leftarrow 0_{n \times m}$
Initialize $c_j \leftarrow 0$ for all $j = 1, \ldots, m$
$t \leftarrow \left\lceil \frac{np}{m} \right\rceil$
Phase 1: Probabilistic balanced assignment
for $i = 1$ to n **do**
    $w_j \leftarrow \max(t - c_j, \epsilon)$ for all j
    Normalize $\{w_j\}$ to probabilities
    Sample p distinct columns $\mathcal{S}_i$
    for $j \in \mathcal{S}_i$ **do**
        $M_{i,j} \leftarrow 1$
        $c_j \leftarrow c_j + 1$
Phase 2: Column coverage repair
for $j = 1$ to m **do**
    $\Delta \leftarrow t - c_j$
    if $\Delta > 0$ **then**
        for $r = 1$ to Δ **do**
            Select row i such that $M_{i,j} = 0$ and $\sum_k M_{i,k} < p$
            $M_{i,j} \leftarrow 1$
            $c_j \leftarrow c_j + 1$
            if $\sum_k M_{i,k} > p$ **then**
                Select k with $M_{i,k} = 1$ and $c_k > t$
                $M_{i,k} \leftarrow 0$
                $c_k \leftarrow c_k - 1$
return M

4.6 Practical Training Considerations

This section discusses practical aspects of training with Subnetwork Data Parallelism (SDP), including compute scaling, fairness in comparison with standard data parallelism, and compatibility with other distributed training techniques.

4.6.1 Compute Scaling and FLOP Matching

Let N denote the total number of parameters and $C \in (0, 1]$ the per-worker parameter density.

Under structured forward masking (e.g., **N-SDP**or **B-SDP**), only the CN active coordinates are materialized during the forward and backward passes. For structured masks applied at the neuron or block level, both activations and arithmetic operations scale approximately proportionally to C. Thus, the per-iteration computational cost satisfies

$$\text{FLOPs}_{\text{SDP}} \approx C \cdot \text{FLOPs}_{\text{DP}}.$$

In backward masking (**$\mathbf{B}_b$-SDP**), the full forward pass is computed, while gradients and optimizer updates are restricted to the CN active coordinates. In this case, forward computation remains unchanged, while backward and optimizer costs scale approximately with C.

When comparing SDP against standard data parallelism, it is therefore important to account for total computational budget. To ensure fair comparisons, one may scale the number of training iterations so that the total floating-point operations are matched across methods. This isolates the effect of masking on optimization behavior from differences in raw compute usage.

4.6.2 Effect of Mask Structure on Stability

The choice of masking granularity influences training stability.

Neuron-Level Masking. In **N-SDP**, masking must be applied consistently across consecutive layers. Removing outputs of layer l induces removal of corresponding inputs in layer $l + 1$. This structural consistency ensures that the masked network defines a valid end-to-end mapping.

Block-Level Masking. In **B-SDP**, entire residual blocks are removed. Because residual architectures preserve identity mappings via skip connections, masking a block does not disrupt dimensional compatibility. This property makes residual networks particularly well-suited for structured block masking.

Backward Masking. Backward masking preserves the full forward representation while reducing gradient storage and synchronization. As shown in the appendix,

backward masking maintains unbiased gradients while introducing a variance inflation term governed by ρ^2. In practice, this variant often exhibits improved stability relative to forward masking at low densities.

4.6.3 Compatibility with Other Parallelisms

SDP operates at the replica level and does not require cross-device activation exchange. Consequently, it is orthogonal to existing model-parallel techniques.

In particular, tensor parallelism, pipeline parallelism, and parameter sharding can be applied within each SDP replica to further reduce memory usage for large models. Conversely, SDP reduces inter-replica communication by restricting synchronization to active coordinates.

This composability allows SDP to be deployed in bandwidth-constrained settings where activation exchange would otherwise dominate communication cost, while still benefiting from intra-replica parallelism for scaling large architectures.

4.6.4 Limiting Cases

Two limiting regimes provide additional intuition:

- When $C = 1$, the masking matrix reduces to the all-ones matrix, and SDP recovers standard Distributed Data Parallel training.
- When masks are disjoint across workers (each coordinate assigned to exactly one worker), synchronization has no effect and training becomes fully partitioned.

These extremes illustrate that SDP interpolates smoothly between full replication and full partitioning, with the overlap structure controlling the trade-off between memory efficiency, communication cost, and statistical efficiency.

Chapter 5

Experimental Evaluation

This chapter empirically evaluates Subnetwork Data Parallelism (SDP) and its structured variants. We validate the theoretical memory and communication reductions derived in Chapter 4, analyze the alignment of masked gradients with full-model updates, and compare optimization performance and efficiency across different masking strategies and parameter densities.

5.1 Experimental Setup

We evaluate Subnetwork Data Parallelism (SDP) across image classification and large language modeling tasks. Experiments are conducted on CIFAR-10 / CIFAR-100 and on large-scale language modeling using the FineWeb dataset.

We define the *parameter density per worker* C as

$$C = \frac{p}{m},$$

where m is the total number of components in the model and p is the number of active components assigned to each of the n workers. Components correspond to computational units such as residual blocks in ResNets or transformer blocks and attention heads in transformer architectures. The density C therefore quantifies the fraction of the model trained on each worker. The case $C = 1$ corresponds to standard data-parallel training.

Structured masks are generated using the balanced construction procedure described in Section 4.5. Unless otherwise stated, hyperparameters are either adopted from prior literature or tuned for the DDP baseline and reused across all SDP variants. This isolates the effect of masking from hyperparameter differences.

5.1.1 ResNet-18 Architecture

We evaluate SDP on ResNet-18 and its wider variant (He et al., 2015; Zagoruyko and Komodakis, 2017) for CIFAR-10 and CIFAR-100 classification.

Training configuration. All experiments use $n = 8$ GPUs with an effective batch size of $\mathcal{B} = 512$, corresponding to 64 samples per GPU. The baseline configuration ($C = 1$) is trained for 200 epochs.

We evaluate the three SDP variants introduced in Chapter 4: **N-SDP**, **B-SDP**, and **B$_b$-SDP**. The number of active blocks per worker is varied as

$$p \in \{8, 7, 6, 5, 4, 3\},$$

corresponding to densities $C = p/8$.

Training uses standard hyperparameters from prior literature, including cosine annealing with linear warm-up over the first 5% of iterations (Goyal et al., 2018; Loshchilov and Hutter, 2017). The peak learning rate is $\eta_{\max} = 0.2$ and the minimum learning rate is $\eta_{\min} = 0.002$. We also evaluate a multi-step schedule where the learning rate is reduced at predefined milestones.

FLOP matching. To ensure fair comparison across densities, we match total computational budget. For **N-SDP** and **B-SDP**, training epochs are scaled inversely with density (e.g., $C = 4/8$ doubles the number of epochs). For **B$_b$-SDP**, we account for the higher backward cost and scale training iterations accordingly (e.g., $C = 4/8$ corresponds to a $1.5\times$ increase in training iterations).

5.1.2 Swin Transformer

We evaluate SDP on Swin-T (Tiny) (Liu et al., 2021), which contains $d = 12$ transformer blocks and follows a hierarchical architecture with attention head counts $\{3, 6, 12, 24\}$ across stages.

Experiments are conducted on CIFAR-10 and CIFAR-100 using $n = 8$ GPUs and an effective batch size of $\mathcal{B} = 512$. The baseline DDP configuration is trained for 400 epochs using the AdamW optimizer with cosine learning rate schedule and 5% warm-up (Loshchilov and Hutter, 2018, 2019). The peak learning rate is $\eta_{\max} = 2 \times 10^{-4}$.

We vary the number of active blocks per worker as

$$p \in \{10, 9, 8, 6, 5\},$$

corresponding to densities $C = p/12$.

For **N-SDP**, structured masking is applied only to attention heads in stages 2–4. Stage 1 is excluded due to its small number of heads, which limits pruning flexibility without significant memory benefit.

As with ResNet experiments, training duration is scaled proportionally to maintain FLOP matching across densities.

5.1.3 Large Language Models

We evaluate SDP on a 134M parameter LLaMA-style transformer trained on the FineWeb dataset (Touvron et al., 2023; Penedo et al., 2024).

The DDP baseline follows Chinchilla scaling laws and is trained with a 3B token budget using 8 workers and a global batch size of 2M tokens (sequence length 1024) (Hoffmann et al., 2022). Optimization uses AdamW with learning rate 8×10^{-3}, weight decay 0.1, and cosine schedule with 10% warm-up.

We compare **N-SDP**, **B-SDP**, and **B$_b$-SDP** under FLOP-matched settings. For **B-SDP** and **B$_b$-SDP**, block-level masking follows the same strategy as in the Swin experiments. For **N-SDP**, structured masks are applied to attention heads in each transformer block.

LLMs impose significant memory constraints in practice, making this setting particularly relevant for evaluating the memory reduction benefits of SDP. All hyperparameters are tuned for the DDP baseline and reused across SDP configurations.

5.2 Empirical Memory and Communication Analysis

Section 4.4.4 derived analytical expressions for the memory footprint of SDP under forward and backward masking. We now empirically validate these reductions using a component-wise decomposition of training-time memory.

5.2.1 Parameter, Gradient, and Optimizer Memory

Figure 3 reports the per-GPU memory footprint for the 134M LLM model under different SDP configurations at coverage ratios that match or outperform the DDP

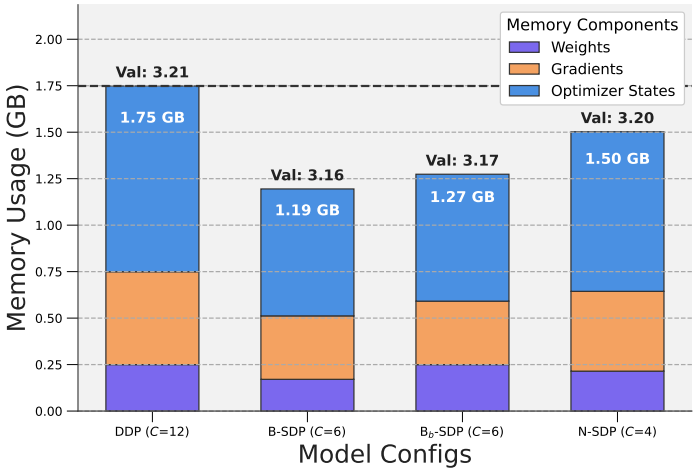


Figure 3: Stacked memory breakdown for different SDP configurations (**B-SDP**, $\mathbf{B}_b$**-SDP**, **N-SDP**) compared to the DDP baseline. Total training memory is decomposed into model weights, gradients, and optimizer states. All SDP variants reduce memory usage while achieving validation loss comparable to or lower than DDP.

baseline. Memory is decomposed into model weights, gradient buffers, and optimizer states.

Compared to DDP, all SDP variants substantially reduce the dominant gradient and optimizer components, while maintaining the full parameter replica on each worker. This confirms the analytical prediction that SDP primarily targets the memory terms that scale with the number of trainable coordinates, rather than static parameter storage.

Under forward masking (**B-SDP** and **N-SDP**), only the active fraction C of coordinates materialize gradient buffers and optimizer accumulators. As derived in Section 4.4.4, this leads to an approximately linear reduction in gradient and optimizer-state memory with respect to C. For example, **B-SDP** at $C = 6/8$ reduces per-GPU memory from 1.75 GB in DDP to 1.19 GB, corresponding to a 32% reduction in total training memory. The majority of savings arise from eliminating inactive optimizer states.

Backward-masked SDP ($\mathbf{B}_b$**-SDP**) exhibits an intermediate memory profile. Although the forward pass materializes the full model, gradient and accumulator storage are restricted to active coordinates. Consequently, optimizer memory scales similarly to **B-SDP**, while gradient memory remains slightly larger due to full forward instantiation. This results in a footprint of 1.27 GB, demonstrating that substantial memory savings can be achieved even when forward computation is unchanged.

Importantly, these reductions are obtained without degrading validation

performance. All SDP configurations match or improve upon the DDP baseline in validation loss, indicating that structured backward sparsification provides a favorable trade-off between memory efficiency and optimization quality.

5.2.2 Activation Memory

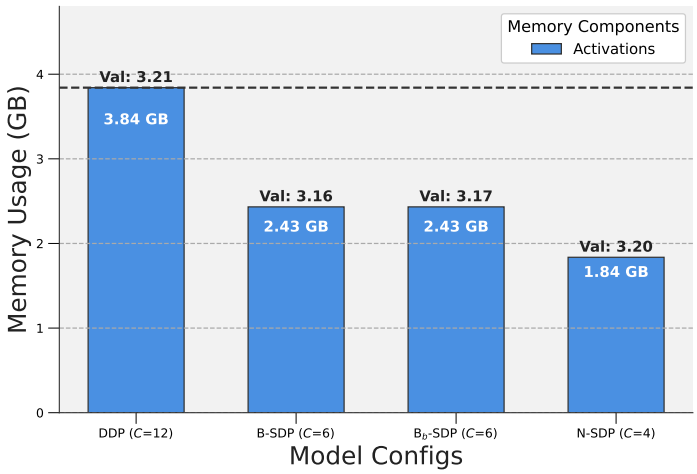


Figure 4: Activation memory footprint for the 134M LLM model under different SDP configurations at coverage ratios achieving validation loss comparable to or better than DDP. Numbers inside bars denote total activation memory (GB); values above indicate validation loss.

Figure 4 compares activation memory across configurations. Standard DDP incurs the highest activation footprint (3.84 GB) with validation loss 3.21. All SDP variants significantly reduce activation memory while maintaining or improving performance.

B-SDP and **B$_b$-SDP** reduce activation memory by approximately 37% (2.43 GB) while achieving lower validation losses (3.16 and 3.17, respectively). **N-SDP** further reduces activation memory to 1.84 GB (a 52% reduction relative to DDP), with validation loss comparable to the baseline.

These results demonstrate that structured masking not only reduces optimizer and gradient storage but also decreases activation memory, enabling substantially more memory-efficient training at comparable model quality.

Overall, these experiments confirm that the theoretical memory scaling derived in Chapter 4 translates directly into practical memory savings across model sizes, without compromising optimization quality.

5.3 Gradient Alignment Analysis

To better understand the optimization behavior of SDP, we analyze the alignment between gradients produced by SDP subnetworks and those of a full-model DDP replica. This analysis provides empirical insight into how masking affects descent directions relative to the true full-model gradient.

5.3.1 Alignment Metric

We measure alignment using cosine similarity. For a given layer or block, let g_{SDP} denote the gradient computed by an SDP configuration and let g_{DDP} denote the gradient from a full DDP replica. The alignment is defined as

$$\cos(g_{\text{SDP}}, g_{\text{DDP}}) = \frac{\langle g_{\text{SDP}}, g_{\text{DDP}} \rangle}{\|g_{\text{SDP}}\| \, \|g_{\text{DDP}}\|}.$$

For **B-SDP**and **B$_b$-SDP**, alignment is computed over active blocks. For **N-SDP**, alignment is computed over active parameters within each layer. In all cases, comparisons are restricted to coordinates that are active under the corresponding mask.

5.3.2 ResNet-18 Analysis

Figure 5 reports gradient alignment across convolutional layers of ResNet-18 at density $C = 4/8$.

Among all methods, **B$_b$-SDP** consistently achieves the highest cosine similarity (approximately 0.6) with the full-model gradient. This behavior aligns with the theoretical analysis shown in the appendix. Because backward masking preserves the full forward computation, the aggregated gradient remains an unbiased estimator of the true gradient, with deviation controlled by the masking operator's distance from uniform averaging. Consequently, **B$_b$-SDP** maintains stronger alignment with the full-model descent direction.

In contrast, **B-SDP** and **N-SDP** exhibit substantially lower alignment, particularly in earlier layers. Under forward masking, the model itself is evaluated on a masked parameter subset, which alters both activations and gradients. This introduces structured deviations from the full-model gradient, leading to reduced cosine similarity.

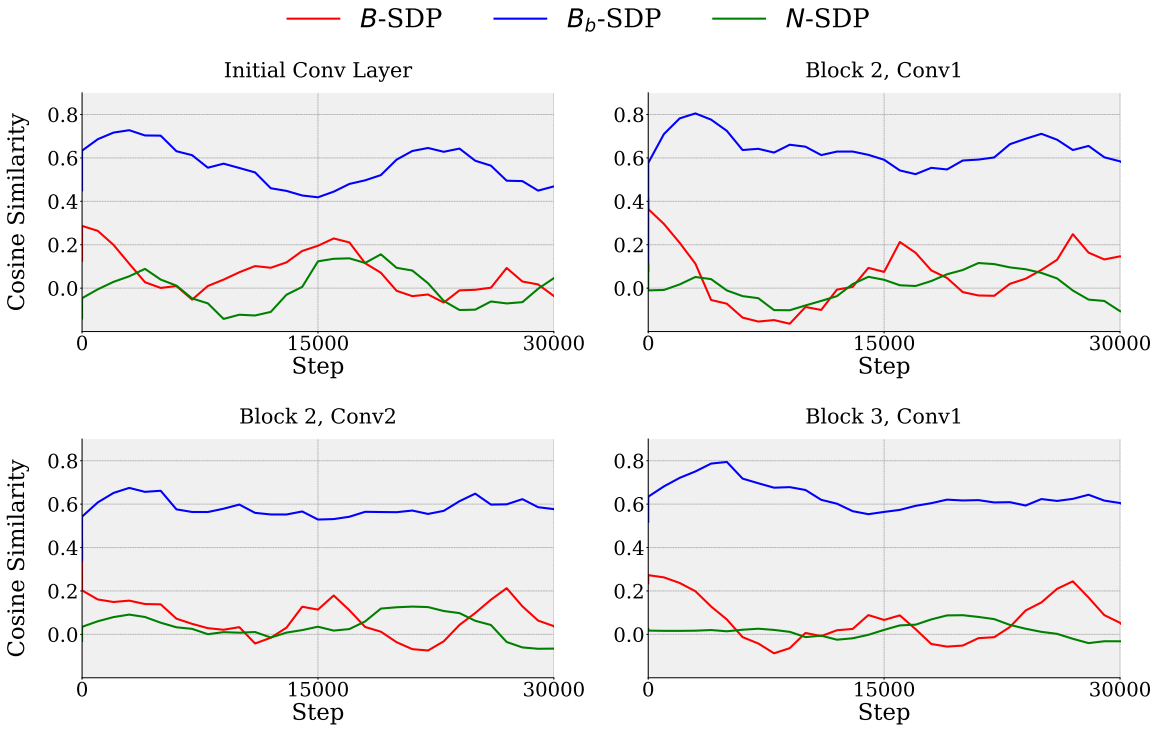


Figure 5: Cosine similarity between gradients produced by SDP subnetworks and the full ResNet-18 model across convolutional layers ($C = 4/8$). **B_b-SDP** achieves the highest alignment with the full-model gradient, while **B-SDP** and **N-SDP** exhibit lower similarity, particularly in early layers.

5.3.3 Interpretation

Interestingly, stronger gradient alignment does not necessarily translate into superior empirical performance. Despite **B_b-SDP** exhibiting the highest cosine similarity, **B-SDP** and **N-SDP** often achieve equal or better validation performance in practice.

This can be explained by two complementary factors:

- Forward-masked variants reduce per-iteration computational cost, enabling longer training under a fixed FLOP budget.
- Even when gradients deviate from the full-model gradient, they may still define effective descent directions for optimizing the global objective.

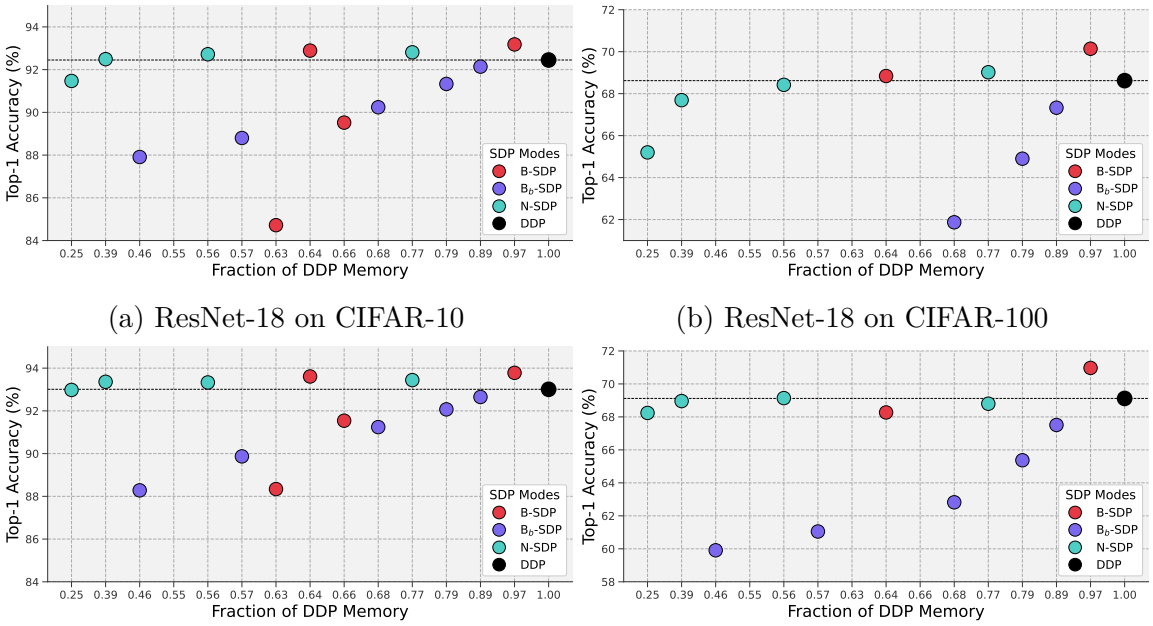
These observations suggest that exact gradient matching is not required for effective optimization under structured subnetwork training. Instead, SDP trades strict alignment for computational efficiency, while preserving sufficiently informative descent directions to maintain or improve performance.

Together with the convergence analysis, these results indicate that backward masking minimizes gradient distortion, whereas forward masking may offer a favorable efficiency–performance trade-off in practice.

5.4 Accuracy Results on Image Classification

We now evaluate the predictive performance of SDP on image classification tasks. In addition to reducing memory footprint (Section 5.2), we examine how parameter density affects accuracy across architectures and masking strategies.

5.4.1 ResNet-18 and Wide-ResNet



(a) ResNet-18 on CIFAR-10

(b) ResNet-18 on CIFAR-100

(c) Wide-ResNet-18 on CIFAR-10

(d) Wide-ResNet-18 on CIFAR-100

Figure 6: Top-1 accuracy (↑) versus fraction of DDP memory for **B-SDP**, **B_b-SDP**, and **N-SDP** on ResNet-18 and WideResNet-18 model across CIFAR-10 and CIFAR100. Each point corresponds to a configuration with a given number of active components determined by coverage ratio C. The black marker and dashed line denote the DDP baseline. Across both architectures and datasets, SDP configurations consistently achieve competitive or higher accuracy while using substantially less memory (**60% reduction**) than the DDP baseline, demonstrating a clear efficiency-performance advantage. In particular, **N-SDP** exhibits the strongest accuracy and memory trade-off, outperforming **B-SDP** and **B_b-SDP** at comparable or lower memory fractions.

Figure 6 summarizes results for ResNet-18 (RN-18) and Wide-ResNet-18 (WRN-18) on CIFAR-10 and CIFAR-100 across varying densities C.

The primary benefit of SDP is the ability to reduce per-worker memory while maintaining competitive accuracy. On CIFAR-10, both RN-18 and WRN-18 retain strong performance even when operating with substantially reduced memory budgets. For example, at approximately 40% of the DDP memory footprint, both architectures achieve competitive accuracy relative to the full-model baseline.

At moderate sparsity levels, forward-masked variants exhibit particularly strong behavior. Using roughly 64% of DDP memory, **N-SDP** surpasses the DDP baseline on both CIFAR-10 and CIFAR-100. This suggests a regularization effect induced by structured subnetwork training, where partial model exposure may improve generalization. Even at approximately 56% of DDP memory, performance remains competitive, especially for **N-SDP**, while offering substantial memory savings.

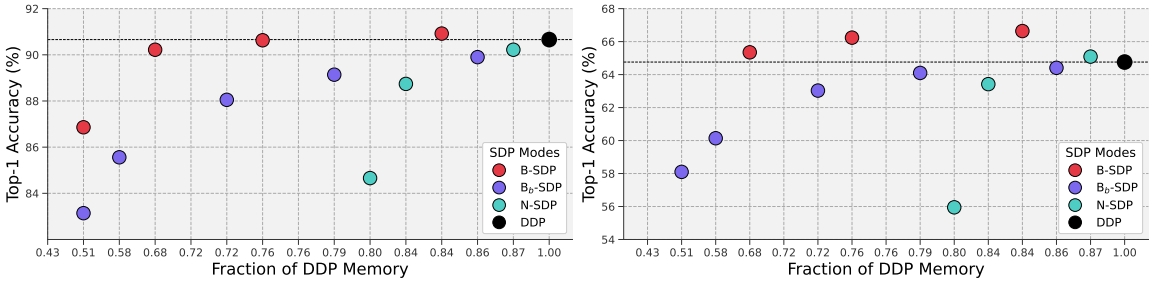
Across both architectures and datasets, performance degrades gracefully as density decreases. WRN-18 demonstrates greater robustness than RN-18 at high sparsity, indicating that wider models may better tolerate structured masking.

Under extreme sparsity (e.g., $C = 3/8$), behavioral differences between masking strategies become more pronounced. In this regime, $\mathbf{B}_b$**-SDP** clearly outperforms **N-SDP** and **B-SDP**, achieving strong accuracy while other variants degrade significantly. This behavior is consistent with the theoretical analysis in Chapter 4: backward masking preserves unbiased gradient estimates and therefore maintains more stable optimization when overlap between workers is minimal. However, this stability comes at the cost of slower per-iteration convergence relative to forward masking.

Two factors help explain the advantage of forward masking at moderate densities:

- Forward masking effectively trains multiple partially distinct subnetworks whose periodic synchronization introduces diversity, which can improve generalization.
- Under FLOP matching, forward-masked variants benefit from increased training iterations relative to backward masking, leading to improved optimization progress.

Finally, repeating experiments with a linear learning rate scheduler yields consistent trends, indicating that the observed behavior is not specific to a particular schedule. Overall, these results demonstrate that SDP provides a favorable memory–accuracy trade-off across convolutional architectures.



(a) Swin-Tiny on CIFAR-10 (b) Swin-Tiny on CIFAR-100

Figure 7: Top-1 accuracy (↑) versus fraction of DDP memory for **B-SDP**, **B_b-SDP**, and **N-SDP** on Swin tiny architecture across CIFAR-10 and CIFAR100. All SDP configurations achieve competitive accuracy while using substantially less memory than the DDP baseline, demonstrating a clear efficiency advantage over full-model training. Notably, unlike convolutional architectures, **B-SDP** consistently delivers the strongest accuracy-memory trade-off on Swin-Tiny (**32% memory reduction**), outperforming **N-SDP** and **B_b-SDP** at comparable or lower memory fractions. This suggests that block-structured subnetworks are particularly well aligned with hierarchical, windowed attention architectures such as Swin.

5.4.2 Swin Transformer

We next evaluate SDP on the Swin-T (Tiny) transformer architecture. Results are shown in Figure 7 for CIFAR-10 and CIFAR-100.

Consistent with the ResNet experiments, **B-SDP** maintains stable performance even as the number of active blocks decreases. On CIFAR-10, accuracy remains near 90% while using approximately 68% of DDP memory. On CIFAR-100, accuracy improves from 64.76% (12 blocks) to 66.64% (10 blocks), representing a 2% gain over the full-model baseline at reduced memory.

N-SDP exhibits different behavior in transformer architectures. While moderate masking is well tolerated, aggressive masking of attention heads leads to a sharp performance drop beyond approximately 80% of DDP memory reduction. This suggests that excessive pruning of attention heads degrades representational capacity in hierarchical transformer models.

As observed for ResNet, **B_b-SDP** maintains the strongest performance under high sparsity, reflecting the benefit of unbiased gradient aggregation in regimes of limited parameter overlap.

Overall, results across CNN and transformer architectures demonstrate that SDP consistently preserves or improves accuracy under substantial memory reductions. The relative performance of masking strategies depends on architecture and sparsity level: forward masking often provides stronger empirical performance at moderate

densities, whereas backward masking offers improved stability under extreme sparsity.

5.5 Large Language Model Results

We now evaluate SDP in large-scale language modeling settings, where memory constraints are particularly acute and training efficiency is critical.

5.5.1 134M Parameter Model

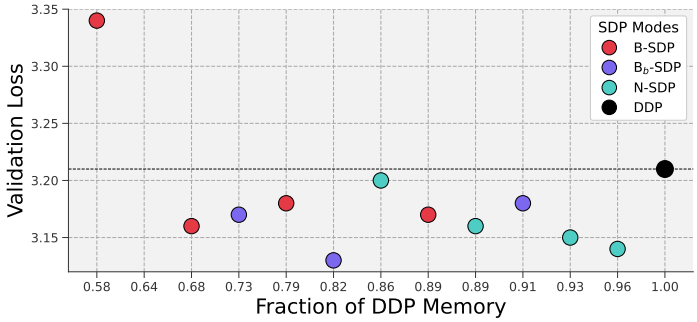


Figure 8: Validation loss (↓) versus fraction of DDP memory for different SDP configurations (**B-SDP**, **B_b-SDP**, **N-SDP**) compared to the DDP baseline. All SDP variants achieve lower validation loss than DDP while using substantially less memory, demonstrating a clear efficiency-performance advantage over full DDP. In particular, **B-SDP** provides the most favorable trade-off, achieving a **32% reduction** in memory footprint while reaching a validation loss of **3.16**, outperforming the DDP baseline (3.21).

We consider a 134M-parameter LLaMA-style transformer trained on the FineWeb dataset. The DDP baseline follows a compute-optimal regime derived from Chinchilla scaling laws, with a total training budget of 3B tokens. All configurations are evaluated under FLOP-matched settings to ensure fair comparison across masking strategies.

We compare **N-SDP**, **B-SDP**, and **B_b-SDP** against standard DDP. For **B-SDP**and **B_b-SDP**, block-level masking follows the same structured design used in the Swin experiments. For **N-SDP**, structured masks are applied to attention heads in each transformer block.

Language models impose substantial memory requirements due to large optimizer states and activation footprints. As shown in Section 5.2, SDP significantly reduces these memory components. We now examine whether these reductions compromise model quality.

Across configurations, SDP consistently matches or improves upon the DDP baseline in validation loss. In particular, **B-SDP** is especially effective: using approximately 68% of the DDP memory footprint, it achieves a validation loss of 3.16 compared to 3.21 for DDP. This demonstrates that structured subnetwork training can yield both improved memory efficiency and improved optimization performance in large-scale autoregressive models.

These results indicate that reducing backward state via structured masking does not degrade the learned language representation. On the contrary, moderate sparsification appears to preserve, and in some cases enhance, generalization performance.

5.5.2 Scaling to 500M Parameters

Table 1: Validation loss (Val. Loss) for different SDP modes comparing them with DDP for a 500M model.

Metric	**DDP** ($C = 1$)	**N-SDP**($C = 8/24$)	**B-SDP**($C = 20/24$)
	2.73	2.69	2.73

To evaluate scalability, we extend the analysis to a 500M-parameter language model trained with approximately 20× more tokens than the compute-optimal Chinchilla budget for the 134M model, following the same scaling principles. Given prior results, we focus on the most promising configurations: **B-SDP** and **N-SDP**.

We evaluate moderate density settings of $C = 20/24$ for **B-SDP** and $C = 8/24$ for **N-SDP**. Results are summarized in Table 1.

Both configurations match or exceed the full-model DDP baseline in validation loss. Notably, **N-SDP** achieves lower validation loss than DDP despite operating at reduced density, demonstrating that the optimization benefits observed at 134M parameters persist at larger scale.

These findings confirm that the memory-efficiency advantages of SDP translate to larger transformer models without sacrificing model quality. Moreover, the stability of performance under moderate sparsification suggests that structured backward-state reduction remains effective even as model capacity and training budgets increase.

Overall, the LLM experiments reinforce the central claim of this work: Subnetwork Data Parallelism enables substantial reductions in per-worker memory while maintaining, and in some regimes improving, optimization performance across both moderate and large-scale language models.

5.6 Discussion and Trade-offs

The experimental results across vision and language models reveal several consistent patterns regarding the trade-offs induced by Subnetwork Data Parallelism (SDP).

5.6.1 Memory–Accuracy Trade-off

Across all architectures, SDP achieves substantial reductions in per-worker memory while maintaining competitive or improved predictive performance. The memory savings arise primarily from eliminating inactive gradient buffers and optimizer states, as predicted by the analytical model in Section 4.4.4. Importantly, these reductions do not require full parameter partitioning or activation exchange across devices.

Performance degrades gracefully as the density C decreases. At moderate sparsity levels, forward-masked variants frequently match or exceed the full-model DDP baseline despite operating with significantly reduced memory. This suggests that structured subnetwork training may act as an implicit regularizer, improving generalization while lowering resource requirements.

Under extreme sparsity, backward masking exhibits greater robustness. This is consistent with the theoretical analysis in the appendix, where backward masking preserves unbiased gradient estimates, with deviation controlled by the mixing operator. As overlap between workers becomes minimal, maintaining unbiased gradient aggregation becomes increasingly important for stability.

5.6.2 Gradient Alignment vs Optimization Quality

The gradient alignment analysis in Section 5.3 showed that $\mathbf{B}_b$**-SDP** maintains the highest cosine similarity with the full-model gradient, reflecting its unbiased backward aggregation. However, stronger alignment does not necessarily translate into superior empirical performance.

Forward-masked variants often outperform backward masking at moderate densities. Two factors help explain this observation:

- Reduced per-iteration computational cost allows forward-masked configurations to train for more iterations under a fixed FLOP budget.
- Even when gradients deviate from the full-model gradient, they may still define effective descent directions for optimizing the global objective.

These results indicate that exact gradient matching is not required for effective optimization. Instead, SDP trades strict gradient alignment for improved computational efficiency, while preserving sufficiently informative descent directions.

5.6.3 Architectural Sensitivity

The relative effectiveness of masking strategies depends on architectural structure.

Convolutional architectures (ResNet and Wide-ResNet) exhibit strong robustness to structured masking, particularly under neuron-level masking. Transformer architectures behave differently: aggressive masking of attention heads can reduce representational capacity, leading to sharper degradation in performance.

Nevertheless, block-level masking remains stable across both CNN and transformer models, and backward masking consistently provides improved stability at high sparsity levels.

5.6.4 Scalability to Large Models

The large language model experiments demonstrate that SDP scales beyond small- and medium-sized models. For both 134M and 500M parameter transformers trained under compute-optimal regimes, SDP matches or exceeds DDP performance while reducing memory requirements.

This confirms that the efficiency gains of SDP are not limited to small architectures. Structured backward-state reduction remains effective even as model size and data scale increase.

5.6.5 Practical Implications

Taken together, these results position SDP as a flexible point in the distributed training design space. It interpolates between fully replicated data parallelism ($C = 1$) and fully partitioned training ($C \to 0$), allowing practitioners to control memory footprint, communication cost, and statistical efficiency through the overlap structure.

Forward masking offers stronger performance at moderate sparsity due to increased training iterations and implicit diversity effects. Backward masking provides improved stability at extreme sparsity due to unbiased gradient aggregation. The choice between variants therefore depends on hardware constraints, desired memory reduction, and acceptable convergence speed.

Overall, SDP enables memory-efficient large-scale training without sacrificing optimization quality, and in several regimes improves generalization relative to standard data parallelism.

Chapter 6

Conclusion

6.1 Summary of Contributions

This thesis introduced *Subnetwork Data Parallelism* (SDP), a distributed training framework designed to address the growing memory bottlenecks in large-scale neural network optimization. Modern training pipelines are increasingly constrained not only by compute but by per-device memory limits arising from parameter replication, gradient buffers, optimizer states, and activation storage. Standard data-parallel training fully replicates these components across workers, while classical model-parallel approaches reduce replication at the cost of increased activation communication. SDP occupies a distinct position in this design space by training structured, overlapping subnetworks on each worker without requiring cross-device activation exchange.

The core contribution of this work is the formalization and empirical validation of this framework. First, we introduced a generic masking abstraction in which each worker is assigned a subset of model coordinates, and synchronization occurs only across overlapping parameters. This formulation allows interpolation between fully replicated data parallelism and fully partitioned training through a controllable density parameter $C \in (0, 1]$.

Second, we proposed structured instantiations of this framework:

- **Neuron-Level SDP (N-SDP)**, which removes neurons or channels in a structured manner across layers;
- **Block-Level SDP (B-SDP)**, which drops entire residual or transformer blocks;

- **Backward-Masked SDP ($\mathbf{B}_b$-SDP)**, which preserves full forward computation while restricting gradient aggregation.

These variants enable practical memory reductions while preserving architectural consistency and end-to-end trainability.

Third, we developed a balanced mask construction algorithm that enforces fixed per-worker budgets while approximately equalizing parameter coverage across workers. This ensures stable synchronization behavior and controlled overlap structure.

Fourth, we provided a theoretical analysis of backward masking under standard L-smoothness assumptions. We showed that masked gradient aggregation preserves unbiasedness while introducing a variance inflation term governed by the distance between the masked mixing operator and uniform averaging. This result formalizes the trade-off between sparsity and convergence rate.

Finally, extensive experiments across convolutional networks, vision transformers, and large language models demonstrated that SDP achieves substantial per-device memory reductions - typically in the range of 32% to 60% while maintaining or even improving predictive performance relative to standard data parallelism. Notably, SDP scales to language models trained under compute-optimal regimes, confirming its practical viability beyond small-scale settings.

Together, these contributions establish Subnetwork Data Parallelism as a flexible and scalable alternative to fully replicated training, enabling larger models and longer sequences to be trained within fixed hardware budgets.

6.2 Empirical Insights

The empirical evaluation of SDP across convolutional networks, vision transformers, and large language models reveals several consistent patterns that clarify the practical behavior of structured subnetwork training.

First, memory reductions closely follow the analytical scaling derived in Chapter 4. Across all architectures, eliminating inactive gradient buffers and optimizer states produces near-linear reductions in per-device memory with respect to the density parameter C. These savings are achieved without requiring parameter partitioning or cross-device activation communication, confirming that SDP effectively targets the dominant memory components in modern training pipelines.

Second, predictive performance degrades gracefully as sparsity increases. At moderate densities, forward-masked variants frequently match or exceed the accuracy

of standard data parallel training. This behavior suggests that structured subnetwork training may introduce an implicit regularization effect, improving generalization while reducing resource usage. The effect is particularly pronounced in convolutional architectures and moderate-scale transformers.

Third, backward masking provides improved stability under extreme sparsity. As overlap between workers becomes minimal, unbiased gradient aggregation becomes increasingly important. Empirically, backward-masked variants maintain stronger performance in high-sparsity regimes, consistent with the theoretical variance analysis presented in Chapter 4.

Fourth, gradient alignment analysis reveals that exact correspondence with the full-model gradient is not required for effective optimization. Although backward masking achieves the highest cosine similarity with the DDP gradient, forward-masked variants often achieve stronger empirical performance under FLOP-matched settings. This indicates that effective descent directions can be maintained even when gradients deviate from the full-model reference, particularly when computational savings allow additional training iterations.

Finally, the language modeling experiments demonstrate that these trends persist at larger scale. For both 134M and 500M parameter models trained under compute-optimal regimes, SDP matches or exceeds DDP performance while operating at reduced memory footprints. This confirms that the observed benefits are not confined to small-scale settings but extend to modern large-model training scenarios.

Overall, the empirical results indicate that SDP offers a controllable trade-off between memory efficiency and optimization fidelity. By adjusting the overlap structure through the density parameter C, practitioners can balance resource constraints against statistical efficiency, enabling scalable training under limited hardware budgets.

6.3 Limitations

While Subnetwork Data Parallelism demonstrates strong empirical performance and substantial memory savings, several limitations should be acknowledged.

First, the theoretical analysis is restricted to the backward-masked setting under standard L-smoothness assumptions. Although this framework provides insight into variance inflation through the mixing operator, it does not capture the full dynamics of forward masking, where gradients are computed on structurally modified models. A complete theoretical characterization of forward-masked optimization behavior

remains an open problem.

Second, hyperparameters were primarily tuned for the DDP baseline and reused across SDP configurations. While this isolates the impact of masking, it may not reflect the optimal training regime for each density level. In practice, density-dependent tuning could further improve performance, particularly under high sparsity.

Third, the effectiveness of structured masking depends on architectural properties. Convolutional networks and residual architectures exhibit strong robustness to block-level masking, whereas aggressive attention-head masking in transformers can degrade representational capacity. The generality of SDP across diverse architectural families therefore depends on the availability of structurally consistent masking schemes.

Fourth, as sparsity increases, the deviation between masked aggregation and uniform averaging grows. Although backward masking preserves unbiasedness, the variance inflation term governed by the mixing operator may slow convergence at very low densities. This introduces a trade-off between memory efficiency and statistical efficiency that may limit extreme sparsification.

Fifth, the communication pattern assumes efficient masked synchronization across overlapping coordinates. While this reduces total communication volume, implementation complexity may increase in large distributed systems where overlapping masks differ across workers. Practical deployment in heterogeneous clusters may require additional engineering considerations.

Finally, this work focuses on structured subnetwork training within fully shared parameter spaces. It does not explore adaptive or learned masking strategies, dynamic overlap schedules, or integration with sparse expert routing mechanisms. These extensions could further improve flexibility and scalability but remain beyond the scope of the present study.

Despite these limitations, the results demonstrate that SDP provides a practical and scalable alternative to fully replicated data parallel training, motivating further investigation into adaptive masking and deeper theoretical analysis.

6.4 Future Work

The results presented in this thesis open several promising directions for future research.

A natural extension concerns *adaptive masking strategies*. In this work, masks are constructed using a balanced assignment procedure and remain fixed throughout

training. Allowing masks to evolve dynamically - either through scheduled density annealing or learned assignment mechanisms - could enable more efficient exploration of the parameter space. For example, early training could employ higher overlap for stability, followed by progressive sparsification to reduce memory footprint.

Another direction involves *learnable or data-dependent mask construction.* Rather than assigning components uniformly, mask selection could incorporate gradient magnitude, activation statistics, or learned importance scores. Such approaches may further improve the trade-off between memory efficiency and optimization quality, particularly under extreme sparsity.

From a theoretical perspective, extending convergence analysis beyond the L-smooth setting remains an important open problem. A deeper understanding of forward-masked dynamics, as well as tighter bounds that characterize the dependence on density C and mixing structure, would strengthen the theoretical foundations of structured subnetwork training.

There is also significant opportunity in combining SDP with other distributed training paradigms. Because SDP operates at the replica level without requiring activation exchange, it is naturally composable with tensor parallelism, pipeline parallelism, and parameter sharding. Exploring joint optimization of overlap structure and intra-replica partitioning could enable training of substantially larger models within fixed hardware constraints.

Finally, structured subnetwork training may interact fruitfully with emerging sparse and modular architectures, including mixture-of-experts models and sparsely activated transformers. Integrating SDP with expert routing or conditional computation mechanisms could further reduce memory and communication costs while preserving expressivity.

Overall, Subnetwork Data Parallelism introduces a flexible framework for trading memory, communication, and statistical efficiency. Continued investigation into adaptive masking, theoretical refinement, and large-scale system integration may further expand its applicability to next-generation large-scale learning systems.

References

Samiul Alam, Luyang Liu, Ming Yan, and Mi Zhang. Fedrolex: Model-heterogeneous federated learning with rolling sub-model extraction. *Advances in neural information processing systems*, 35:29677–29690, 2022. 19, 20, 21

Léon Bottou. Large-scale machine learning with stochastic gradient descent. In *Proceedings of COMPSTAT'2010: 19th International Conference on Computational StatisticsParis France, August 22-27, 2010 Keynote, Invited and Contributed Papers*, pages 177–186. Springer, 2010. 63

Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners, 2020. URL `https://arxiv.org/abs/2005.14165`. 2

Sebastian Caldas, Jakub Konečny, H Brendan McMahan, and Ameet Talwalkar. Expanding the reach of federated learning by reducing client resource requirements. *arXiv preprint arXiv:1812.07210*, 2018. 19

Sebastian Caldas et al. Federated learning with matched averaging. In *International Conference on Learning Representations*, 2019. 21

DeepSeek-AI, Aixin Liu, Bei Feng, Bing Xue, Bingxuan Wang, Bochao Wu, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenyu Zhang, Chong Ruan, Damai Dai, Daya Guo, Dejian Yang, Deli Chen, Dongjie Ji, Erhang Li, Fangyun Lin, Fucong Dai, Fuli Luo, Guangbo Hao, Guanting Chen, Guowei Li, H. Zhang, Han Bao, Hanwei Xu, Haocheng Wang, Haowei Zhang, Honghui Ding, Huajian Xin, Huazuo

Gao, Hui Li, Hui Qu, J. L. Cai, Jian Liang, Jianzhong Guo, Jiaqi Ni, Jiashi Li, Jiawei Wang, Jin Chen, Jingchang Chen, Jingyang Yuan, Junjie Qiu, Junlong Li, Junxiao Song, Kai Dong, Kai Hu, Kaige Gao, Kang Guan, Kexin Huang, Kuai Yu, Lean Wang, Lecong Zhang, Lei Xu, Leyi Xia, Liang Zhao, Litong Wang, Liyue Zhang, Meng Li, Miaojun Wang, Mingchuan Zhang, Minghua Zhang, Minghui Tang, Mingming Li, Ning Tian, Panpan Huang, Peiyi Wang, Peng Zhang, Qiancheng Wang, Qihao Zhu, Qinyu Chen, Qiushi Du, R. J. Chen, R. L. Jin, Ruiqi Ge, Ruisong Zhang, Ruizhe Pan, Runji Wang, Runxin Xu, Ruoyu Zhang, Ruyi Chen, S. S. Li, Shanghao Lu, Shangyan Zhou, Shanhuang Chen, Shaoqing Wu, Shengfeng Ye, Shengfeng Ye, Shirong Ma, Shiyu Wang, Shuang Zhou, Shuiping Yu, Shunfeng Zhou, Shuting Pan, T. Wang, Tao Yun, Tian Pei, Tianyu Sun, W. L. Xiao, Wangding Zeng, Wanjia Zhao, Wei An, Wen Liu, Wenfeng Liang, Wenjun Gao, Wenqin Yu, Wentao Zhang, X. Q. Li, Xiangyue Jin, Xianzu Wang, Xiao Bi, Xiaodong Liu, Xiaohan Wang, Xiaojin Shen, Xiaokang Chen, Xiaokang Zhang, Xiaosha Chen, Xiaotao Nie, Xiaowen Sun, Xiaoxiang Wang, Xin Cheng, Xin Liu, Xin Xie, Xingchao Liu, Xingkai Yu, Xinnan Song, Xinxia Shan, Xinyi Zhou, Xinyu Yang, Xinyuan Li, Xuecheng Su, Xuheng Lin, Y. K. Li, Y. Q. Wang, Y. X. Wei, Y. X. Zhu, Yang Zhang, Yanhong Xu, Yanhong Xu, Yanping Huang, Yao Li, Yao Zhao, Yaofeng Sun, Yaohui Li, Yaohui Wang, Yi Yu, Yi Zheng, Yichao Zhang, Yifan Shi, Yiliang Xiong, Ying He, Ying Tang, Yishi Piao, Yisong Wang, Yixuan Tan, Yiyang Ma, Yiyuan Liu, Yongqiang Guo, Yu Wu, Yuan Ou, Yuchen Zhu, Yuduan Wang, Yue Gong, Yuheng Zou, Yujia He, Yukun Zha, Yunfan Xiong, Yunxian Ma, Yuting Yan, Yuxiang Luo, Yuxiang You, Yuxuan Liu, Yuyang Zhou, Z. F. Wu, Z. Z. Ren, Zehui Ren, Zhangli Sha, Zhe Fu, Zhean Xu, Zhen Huang, Zhen Zhang, Zhenda Xie, Zhengyan Zhang, Zhewen Hao, Zhibin Gou, Zhicheng Ma, Zhigang Yan, Zhihong Shao, Zhipeng Xu, Zhiyu Wu, Zhongyu Zhang, Zhuoshu Li, Zihui Gu, Zijia Zhu, Zijun Liu, Zilin Li, Ziwei Xie, Ziyang Song, Ziyi Gao, and Zizheng Pan. Deepseek-v3 technical report, 2025. URL `https://arxiv.org/abs/2412.19437`. 9, 16

Enmao Diao, Jie Ding, and Vahid Tarokh. Heterofl: Computation and communication efficient federated learning for heterogeneous clients. *ICLR*, 2021. 20

Arthur Douillard, Qixuan Feng, Andrei A Rusu, Rachita Chhaparia, Yani Donchev, Adhiguna Kuncoro, Marc'Aurelio Ranzato, Arthur Szlam, and Jiajun Shen. Diloco: Distributed low-communication training of language models. *arXiv preprint arXiv:2311.08105*, 2023. 20

Erwan Fagnou, Paul Caillon, Blaise Delattre, and Alexandre Allauzen. Accelerated training through iterative gradient propagation along the residual path. *arXiv preprint arXiv:2501.17086*, 2025. 20

Priya Goyal, Piotr Dollár, Ross Girshick, Pieter Noordhuis, Lukasz Wesolowski, Aapo Kyrola, Andrew Tulloch, Yangqing Jia, and Kaiming He. Accurate, large minibatch sgd: Training imagenet in 1 hour, 2018. 8, 36

Dhruv Guliani, Lillian Zhou, Changwan Ryu, Tien-Ju Yang, Harry Zhang, Yonghui Xiao, Françoise Beaufays, and Giovanni Motta. Enabling on-device training of speech recognition models with federated dropout. In *ICASSP 2022-2022 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 8757–8761. IEEE, 2022. 19, 21

Aaron Harlap, Deepak Narayanan, Amar Phanishayee, Vivek Seshadri, Nikhil Devanur, Greg Ganger, and Phil Gibbons. Pipedream: Fast and efficient pipeline parallel dnn training, 2018. URL `https://arxiv.org/abs/1806.03377`. 2, 9

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition, 2015. URL `https://arxiv.org/abs/1512.03385`. 36

Jordan Hoffmann, Sebastian Borgeaud, Arthur Mensch, Elena Buchatskaya, Trevor Cai, Eliza Rutherford, Diego de Las Casas, Lisa Anne Hendricks, Johannes Welbl, Aidan Clark, et al. Training compute-optimal large language models. *arXiv preprint arXiv:2203.15556*, 2022. 37

Samuel Horvath, Stefanos Laskaridis, Mario Almeida, Ilias Leontiadis, Stylianos Venieris, and Nicholas Lane. Fjord: Fair and accurate federated learning under heterogeneous targets with ordered dropout. *Advances in Neural Information Processing Systems*, 34:12876–12889, 2021. 19, 21

Yanping Huang, Youlong Cheng, Ankur Bapna, Orhan Firat, Mia Xu Chen, Dehao Chen, HyoukJoong Lee, Jiquan Ngiam, Quoc V. Le, Yonghui Wu, and Zhifeng Chen. Gpipe: Efficient training of giant neural networks using pipeline parallelism, 2019. 2, 9, 15

Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B. Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. Scaling laws for neural language models, 2020. URL `https://arxiv.org/abs/2001.08361`. 2

Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014. 11

Jakub Konečný, H. Brendan McMahan, Daniel Ramage, and Peter RichtÃąrik. Federated optimization: Distributed machine learning for on-device intelligence, 2016. URL https://arxiv.org/abs/1610.02527. 19, 21

Shen Li, Yanli Zhao, Rohan Varma, Omkar Salpekar, Pieter Noordhuis, Teng Li, Adam Paszke, Jeff Smith, Brian Vaughan, Pritam Damania, and Soumith Chintala. Pytorch distributed: Experiences on accelerating data parallel training, 2020. URL https://arxiv.org/abs/2006.15704. 2

Ze Liu, Yutong Lin, Yue Cao, Han Hu, Yixuan Wei, Zheng Zhang, Stephen Lin, and Baining Guo. Swin transformer: Hierarchical vision transformer using shifted windows, 2021. URL https://arxiv.org/abs/2103.14030. 36

Ilya Loshchilov and Frank Hutter. Sgdr: Stochastic gradient descent with warm restarts, 2017. URL https://arxiv.org/abs/1608.03983. 36

Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. In *International Conference on Learning Representations*, 2018. URL https://arxiv.org/abs/1711.05101. 36

Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net, 2019. URL https://openreview.net/forum?id=Bkg6RiCqY7. 36

Paulius Micikevicius, Sharan Narang, Jonah Alben, Gregory Diamos, Erich Elsen, David Garcia, Boris Ginsburg, Michael Houston, Oleksii Kuchaiev, Ganesh Venkatesh, and Hao Wu. Mixed precision training, 2018. URL https://arxiv.org/abs/1710.03740. 11, 30

Pitch Patarasuk and Xin Yuan. Bandwidth optimal all-reduce algorithms for clusters of workstations. *J. Parallel Distrib. Comput.*, 69(2):117–124, February 2009. ISSN 0743-7315. doi: 10.1016/j.jpdc.2008.09.002. URL https://doi.org/10.1016/j.jpdc.2008.09.002. 22, 31

Guilherme Penedo, Hynek Kydlíček, Loubna Ben allal, Anton Lozhkov, Margaret Mitchell, Colin Raffel, Leandro Von Werra, and Thomas Wolf. The fineweb

datasets: Decanting the web for the finest text data at scale, 2024. URL https://arxiv.org/abs/2406.17557. 37

Samyam Rajbhandari, Jeff Rasley, Olatunji Ruwase, and Yuxiong He. Zero: memory optimizations toward training trillion parameter models. In Christine Cuicchi, Irene Qualters, and William T. Kramer, editors, *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, SC 2020, Virtual Event / Atlanta, Georgia, USA, November 9-19, 2020*, page 20. IEEE/ACM, 2020. 10, 30

Sashank J. Reddi, Zachary Charles, Manzil Zaheer, Zachary Garrett, Keith Rush, Jakub Konečný, Sanjiv Kumar, and Hugh Brendan McMahan. Adaptive federated optimization. In *International Conference on Learning Representations*, 2021. URL https://openreview.net/forum?id=LkFG3lB13U5. 20

Max Ryabinin, Tim Dettmers, Michael Diskin, and Alexander Borzunov. Swarm parallelism: Training large models can be surprisingly communication-efficient. In *International Conference on Machine Learning*, pages 29416–29440. PMLR, 2023. 18

Noam Shazeer, Youlong Cheng, Niki Parmar, Dustin Tran, Ashish Vaswani, Penporn Koanantakool, Peter Hawkins, HyoukJoong Lee, Mingsheng Hong, Cliff Young, et al. Mesh-tensorflow: Deep learning for supercomputers. *Advances in neural information processing systems*, 31, 2018. 10

Mohammad Shoeybi et al. Megatron-lm: Training multi-billion parameter language models using model parallelism. *arXiv preprint arXiv:1909.08053*, 2019. 10

Zhenheng Tang, Shaohuai Shi, Wei Wang, Bo Li, and Xiaowen Chu. Communication-efficient distributed deep learning: A comprehensive survey, 2023. URL https://arxiv.org/abs/2003.06307. 22, 31

Kimi Team, Yifan Bai, Yiping Bao, Y. Charles, Cheng Chen, Guanduo Chen, Haiting Chen, Huarong Chen, Jiahao Chen, Ningxin Chen, Ruijue Chen, Yanru Chen, Yuankun Chen, Yutian Chen, Zhuofu Chen, Jialei Cui, Hao Ding, Mengnan Dong, Angang Du, Chenzhuang Du, Dikang Du, Yulun Du, Yu Fan, Yichen Feng, Kelin Fu, Bofei Gao, Chenxiao Gao, Hongcheng Gao, Peizhong Gao, Tong Gao, Yuyao Ge, Shangyi Geng, Qizheng Gu, Xinran Gu, Longyu Guan, Haiqing Guo, Jianhang Guo, Xiaoru Hao, Tianhong He, Weiran He, Wenyang He, Yunjia He, Chao Hong,

Hao Hu, Yangyang Hu, Zhenxing Hu, Weixiao Huang, Zhiqi Huang, Zihao Huang, Tao Jiang, Zhejun Jiang, Xinyi Jin, Yongsheng Kang, Guokun Lai, Cheng Li, Fang Li, Haoyang Li, Ming Li, Wentao Li, Yang Li, Yanhao Li, Yiwei Li, Zhaowei Li, Zheming Li, Hongzhan Lin, Xiaohan Lin, Zongyu Lin, Chengyin Liu, Chenyu Liu, Hongzhang Liu, Jingyuan Liu, Junqi Liu, Liang Liu, Shaowei Liu, T. Y. Liu, Tianwei Liu, Weizhou Liu, Yangyang Liu, Yibo Liu, Yiping Liu, Yue Liu, Zhengying Liu, Enzhe Lu, Haoyu Lu, Lijun Lu, Yashuo Luo, Shengling Ma, Xinyu Ma, Yingwei Ma, Shaoguang Mao, Jie Mei, Xin Men, Yibo Miao, Siyuan Pan, Yebo Peng, Ruoyu Qin, Zeyu Qin, Bowen Qu, Zeyu Shang, Lidong Shi, Shengyuan Shi, Feifan Song, Jianlin Su, Zhengyuan Su, Lin Sui, Xinjie Sun, Flood Sung, Yunpeng Tai, Heyi Tang, Jiawen Tao, Qifeng Teng, Chaoran Tian, Chensi Wang, Dinglu Wang, Feng Wang, Hailong Wang, Haiming Wang, Jianzhou Wang, Jiaxing Wang, Jinhong Wang, Shengjie Wang, Shuyi Wang, Si Wang, Xinyuan Wang, Yao Wang, Yejie Wang, Yiqin Wang, Yuxin Wang, Yuzhi Wang, Zhaoji Wang, Zhengtao Wang, Zhengtao Wang, Zhexu Wang, Chu Wei, Qianqian Wei, Haoning Wu, Wenhao Wu, Xingzhe Wu, Yuxin Wu, Chenjun Xiao, Jin Xie, Xiaotong Xie, Weimin Xiong, Boyu Xu, Jinjing Xu, L. H. Xu, Lin Xu, Suting Xu, Weixin Xu, Xinran Xu, Yangchuan Xu, Ziyao Xu, Jing Xu, Jing Xu, Junjie Yan, Yuzi Yan, Hao Yang, Xiaofei Yang, Yi Yang, Ying Yang, Zhen Yang, Zhilin Yang, Zonghan Yang, Haotian Yao, Xingcheng Yao, Wenjie Ye, Zhuorui Ye, Bohong Yin, Longhui Yu, Enming Yuan, Hongbang Yuan, Mengjie Yuan, Siyu Yuan, Haobing Zhan, Dehao Zhang, Hao Zhang, Wanlu Zhang, Xiaobin Zhang, Yadong Zhang, Yangkun Zhang, Yichi Zhang, Yizhi Zhang, Yongting Zhang, Yu Zhang, Yutao Zhang, Yutong Zhang, Zheng Zhang, Haotian Zhao, Yikai Zhao, Zijia Zhao, Huabin Zheng, Shaojie Zheng, Longguang Zhong, Jianren Zhou, Xinyu Zhou, Zaida Zhou, Jinguo Zhu, Zhen Zhu, Weiyu Zhuang, and Xinxing Zu. Kimi k2: Open agentic intelligence, 2026. URL `https://arxiv.org/abs/2507.20534`. 2

Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, et al. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*, 2023. URL `https://arxiv.org/abs/2302.13971`. 37

Jue Wang, Yucheng Lu, Binhang Yuan, Beidi Chen, Percy Liang, Christopher De Sa, Christopher Re, and Ce Zhang. Cocktailsgd: Fine-tuning foundation models over 500mbps networks. In *International Conference on Machine Learning*, pages 36058–36076. PMLR, 2023. 20

Dingzhu Wen, Ki-Jun Jeon, and Kaibin Huang. Federated dropout - a simple approach for enabling federated learning on resource constrained devices. *IEEE wireless communications letters*, 11(5):923–927, 2022. ix, 19, 21

Binhang Yuan, Cameron R Wolfe, Chen Dun, Yuxin Tang, Anastasios Kyrillidis, and Christopher M Jermaine. Distributed learning of deep neural networks using independent subnet training. *arXiv preprint arXiv:1910.02120*, 2019. 20

Sergey Zagoruyko and Nikos Komodakis. Wide residual networks, 2017. URL `https://arxiv.org/abs/1605.07146`. 36

Yanli Zhao, Andrew Gu, Rohan Varma, Liang Luo, Chien-Chin Huang, Min Xu, Less Wright, Hamid Shojanazeri, Myle Ott, Sam Shleifer, Alban Desmaison, Can Balioglu, Pritam Damania, Bernard Nguyen, Geeta Chauhan, Yuchen Hao, Ajit Mathews, and Shen Li. Pytorch fsdp: Experiences on scaling fully sharded data parallel, 2023. URL `https://arxiv.org/abs/2304.11277`. 10

Appendix

A Theoretical Analysis

We now analyze the convergence behavior of Subnetwork Data Parallelism under *backward masking*. In this setting, the forward pass uses the full model while masking is applied only during gradient aggregation.

A.1 Setting and Assumptions

Let $f : \mathbb{R}^{|J|} \to \mathbb{R}$ denote the objective function. We assume:

- f is L-smooth, i.e.,

$$f(y) \leq f(x) + \langle \nabla f(x), y - x \rangle + \frac{L}{2}\|y - x\|^2 \quad \forall x, y.$$

- Each worker computes a stochastic gradient g_i^t at $\boldsymbol{\theta}^t$ satisfying

$$\mathbb{E}[g_i^t \mid \boldsymbol{\theta}^t] = \nabla f(\boldsymbol{\theta}^t),$$

$$\mathbb{E}\Big[\|g_i^t - \nabla f(\boldsymbol{\theta}^t)\|^2 \mid \boldsymbol{\theta}^t\Big] \leq \sigma^2.$$

Under backward masking, gradients are aggregated via the masked averaging operator:

$$\widehat{g}^t = \bar{m}(g_1^t, \ldots, g_n^t).$$

Let the uniform averaging operator be

$$g_{\text{uni}}^t = \bar{m}^{\text{uni}}(g_1^t, \ldots, g_n^t) = \frac{1}{n}\sum_{i=1}^{n} g_i^t.$$

Define the masking deviation

$$\delta^t = \widehat{g}^t - g_{\text{uni}}^t.$$

The parameter update is

$$\boldsymbol{\theta}^{t+1} = \boldsymbol{\theta}^t - \eta \widehat{g}^t.$$

A.2 Variance Under Linear Mixing

It is a well known result Bottou (2010) that if f is L-smooth and $\eta \leq \frac{1}{2L}$, then for any $T \geq 1$,

$$\frac{1}{T}\sum_{t=0}^{T-1}\mathbb{E}\|\nabla f(\boldsymbol{\theta}^t)\|^2 \leq \frac{2(f(\boldsymbol{\theta}^0)-f^\star)}{\eta T} + L\eta\,\sigma^2\rho^2. \tag{3}$$

We also require the following variance identity for linear aggregation.

[Variance under linear mixing] Let $x_1, \ldots, x_m \in \mathbb{R}^n$ be i.i.d. with $\mathbb{E}[x_i] = x$ and $\mathrm{Cov}(x_i) = \sigma^2 I_n$. Define the sample mean $\bar{x} \coloneqq \frac{1}{m}\sum_{i=1}^m x_i \in \mathbb{R}^n$ and let

$$P : (\mathbb{R}^n)^m \to \mathbb{R}^n, \qquad (Px)_j \coloneqq \sum_{i=1}^m P_{ij}\,(x_i)_j, \quad j = 1, \ldots, n.$$

Then

$$\mathbb{E}\,\|Px - \bar{x}\|_2^2 = \sigma^2 \Big\| P - \frac{1}{m}\mathbf{1}_m\mathbf{1}_n^\top \Big\|_F^2. \tag{4}$$

Proof. For each coordinate $j \in \{1, \ldots, n\}$,

$$(Px - \bar{x})_j = \sum_{i=1}^m \Big(P_{ij} - \frac{1}{m}\Big)(x_i)_j.$$

By independence across i and the assumption $\mathrm{Var}((x_i)_j) = \sigma^2$,

$$\begin{aligned}
\mathbb{E}\,\|Px - \bar{x}\|_2^2 &= \sum_{j=1}^n \mathbb{E}\,(Px - \bar{x})_j^2 = \sum_{j=1}^n \mathrm{Var}\left(\sum_{i=1}^m \Big(P_{ij} - \frac{1}{m}\Big)(x_i)_j\right) \\
&= \sum_{j=1}^n \sum_{i=1}^m \Big(P_{ij} - \frac{1}{m}\Big)^2 \mathrm{Var}((x_i)_j) = \sigma^2 \sum_{j=1}^n \sum_{i=1}^m \Big(P_{ij} - \frac{1}{m}\Big)^2 \\
&= \sigma^2 \Big\| P - \frac{1}{m}\mathbf{1}_m\mathbf{1}_n^\top \Big\|_F^2,
\end{aligned}$$

which concludes the proof. □

Consequence for equation 3. By Lemma A.2, the effective variance term induced by the linear mixing operator P is $\sigma^2 \| P - \frac{1}{m}\mathbf{1}_m\mathbf{1}_n^\top \|_F^2$. Therefore, replacing σ^2 in equation 3 by this quantity yields the desired bound of Theorem A.3.

A.3 Application to Backward Masking

In the backward-masking setting, the masked aggregation operator induces a linear mixing operator P over workers (take $m = n$ in Lemma A.2). The corresponding

deviation from uniform averaging is measured by

$$\rho^2 = \left\| \frac{1}{n} m^{\text{uni}} - \frac{1}{c} m \right\|_F^2,$$

which is the Frobenius distance to the uniform masking.

[SGD rate under Backward-masking] Assume f is L-smooth and choose

$$\eta \leq \frac{1}{2L(1+n\rho^2)}.$$

Then for any $T \geq 1$,

$$\frac{1}{T}\sum_{t=0}^{T-1} \mathbb{E}\|\nabla f(\boldsymbol{\theta}^t)\|^2 \leq \frac{2(f(\boldsymbol{\theta}^0) - f^\star)}{\eta T} + L\eta\sigma^2\rho^2,$$

where $f^\star \coloneqq \inf_{\boldsymbol{\theta}} f(\boldsymbol{\theta})$.

A.4 Interpretation

The bound matches classical SGD up to a multiplicative variance inflation term ρ^2, which quantifies how far the masked aggregation operator deviates from uniform averaging. When the masked mixing reduces to uniform averaging, we have $\rho = 0$ and the bound reduces to standard data-parallel SGD. As the masking deviates further from uniform assignment, the effective variance increases proportionally to ρ^2, slowing convergence but preserving the same asymptotic rate structure.