



# Colossal-AI: A Unified Deep Learning System For Large-Scale Parallel Training

Shenggui Li  
lisg@hpcaitech.com  
HPC-AI Technology Inc.  
Singapore

Hongxin Liu  
liuhongxin@hpcaitech.com  
HPC-AI Technology Inc.  
Beijing, China

Zhengda Bian  
bian.zhengda@hpcaitech.com  
HPC-AI Technology Inc.  
Beijing, China

Jiarui Fang  
fangjiarui123@gmail.com  
HPC-AI Technology Inc.  
Beijing, China

Haichen Huang  
haichen.cs@hotmail.com  
HPC-AI Technology Inc.  
Beijing, China

Yuliang Liu  
yuliangliu888@gmail.com  
HPC-AI Technology Inc.  
Beijing, China

Boxiang Wang  
bwang@g.harvard.edu  
Singapore HPC-AI Technology Inc.

Yang You\*  
youy@comp.nus.edu.sg  
National University of Singapore  
Singapore

## ABSTRACT

The success of Transformer models has pushed the deep learning model scale to billions of parameters, but the memory limitation of a single GPU has led to an urgent need for training on multi-GPU clusters. However, the best practice for choosing the optimal parallel strategy is still lacking, as it requires domain expertise in both deep learning and parallel computing. The Colossal-AI system addressed the above challenge by introducing a unified interface to scale your sequential code of model training to distributed environments. It supports parallel training methods such as data, pipeline, tensor, and sequence parallelism and is integrated with heterogeneous training and zero redundancy optimizer. Compared to the baseline system, Colossal-AI can achieve up to 2.76 times training speedup on large-scale models.

## CCS CONCEPTS

• Computing methodologies → Machine learning algorithms; Parallel computing methodologies.

## KEYWORDS

datasets, neural networks, gaze detection, text tagging

### ACM Reference Format:

Shenggui Li, Hongxin Liu, Zhengda Bian, Jiarui Fang, Haichen Huang, Yuliang Liu, Boxiang Wang, and Yang You. 2023. Colossal-AI: A Unified Deep Learning System For Large-Scale Parallel Training. In *52nd International Conference on Parallel Processing (ICPP 2023)*, August 07–10, 2023, Salt Lake City, UT, USA.

\*Yang You is a faculty member at NUS, this work was done at HPC-AI Technology Inc.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from [permissions@acm.org](https://permissions.acm.org).

ICPP 2023, August 07–10, 2023, Salt Lake City, UT, USA

© 2023 Copyright held by the owner/author(s). Publication rights licensed to ACM.  
ACM ISBN 979-8-4007-0843-5/23/08...\$15.00  
<https://doi.org/10.1145/3605573.3605613>

City, UT, USA. ACM, New York, NY, USA, 10 pages. <https://doi.org/10.1145/3605573.3605613>

## 1 INTRODUCTION

Deep learning has been successful in many applications and brought breakthroughs in difficult problems. With large amounts of data, neural networks like BERT [8] and Vision Transformer [9] are capable of learning high-dimensional features and making predictions on a level even humans cannot match. As powerful hardware becomes available, neural networks have more diverse architectures and a larger number of parameters. The AI community has seen a trend of deep learning models becoming larger, with an array of large-scale models ranging from BERT-Large, GPT-2 [28] (1.5 billion parameters), GPT-3 [5] (175 billion parameters), to GLM [10] (1.75 trillion parameters). These large-scale models require more data and computing resources but also have better generality and performance. As more robust computing hardware and larger datasets become available, the trend is expected to continue and traditional training methods will become less effective, making distributed training necessary for large-scale model training.

The limited fast memory of commonly used accelerator hardware, such as GPU, is a bottleneck of scaling the model to billions of parameters. The memory consumption in deep learning comes from model parameters, layer activations, gradients, and optimizer states. We refer to model parameters, gradients, and optimizer states as model data and layer activations as non-model data. When training with adaptive optimizers [11, 18], the total memory consumption of model data can be several times larger than that consumed by parameters alone, making a single GPU no longer sufficient for large-scale model training. 10 billion parameters in FP16 format can already consume 20 GB of model memory, while a typical GPU only has 16 or 32 GB of memory. Without any optimization, training a model of 10 billion parameters with one data sample can cost more than 80 GB of memory, which is far more than that of a typical GPU.

Data parallelism [31] has scaled models such as ResNet [14] to multi-GPU training and other methods such as activation checkpointing [7] were proposed to reduce the non-model data by trading computation for memory. However, these methods failed to cope with billion-parameter model data. Parallelization techniques such as pipeline parallelism [15, 24] and tensor parallelism [32] were explored to shard the model data, making it possible to train models at a larger scale. The current state-of-the-art systems which provide a solution to the scaling challenge include GShard [19], FairScale [2], Megatron-LM [26] and DeepSpeed [29]. Among these systems, Megatron-LM and DeepSpeed are the most popular in the open-source community and deliver the best performance. Thus, they are chosen as the baseline of our experiments. Megatron-LM trains Transformer-based models by utilizing optimized pipeline and tensor parallelism. Meanwhile, DeepSpeed proposed an efficient method to partition the model-related data to fully eliminate memory redundancy in data parallel training. These two efficient methods paved the way to scale model training to hundreds of devices and billions of parameters.

As most deep learning engineers and researchers are used to writing non-distributed code, it is reasonably difficult for them to adapt to parallel and distributed programming. The existing systems either introduce extra complexity in parallelizing the model training or offer insufficient parallelization techniques. We have thus developed Colossal-AI, which is an open-source system to democratize complicated distributed training in the AI community by unifying an array of training acceleration techniques in one deep learning system. In this system, we also included novel parallelism methods such as multi-dimension tensor parallelism and sequence parallelism. Colossal-AI aims to make distributed training easy by providing user-friendly APIs while allowing users to maintain their coding habit of writing single-node programs. In a nutshell, we bring the following major contributions to large-scale distributed training in this work:

- Colossal-AI is a unified deep learning system that provides the fullest set of acceleration techniques for the AI community. With its modular design as shown in Figure 1, Colossal-AI allows for free combination of these techniques to achieve the best training speedup. The details of the system architecture will be discussed in the Implementation section.
- Optimized parallelism and heterogeneous training methods are provided in Colossal-AI. These methods achieve better system performance than the baseline systems. They are provided for the user via friendly APIs with minimum code changes.
- In-depth analysis was conducted to investigate the suitable parallelism strategies under different hardware conditions.

## 2 BACKGROUND

Thanks to the advent of the Transformer architecture, deep learning models have gained unprecedented performance improvement in domains such as Computer Vision and Natural Language Processing. The typical architecture of a Transformer layer consists of a Multi-head Attention block and a Feed Forward block as shown in Figure 2. This architecture can scale to billions of parameters and

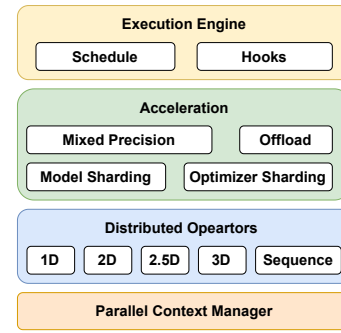


Figure 1: Architecture of Colossal-AI

larger models can deliver more impressive performance improvement. For example, GPT-3 [5] outperforms the smaller models by 18% absolute increase in prediction accuracy on the LAMBADA language task [27].

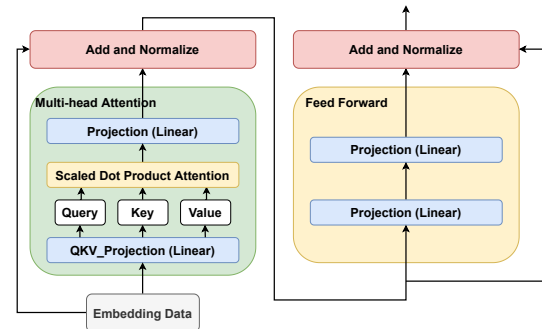


Figure 2: Architecture of The Transformer Layer

To cope with the increasing model size, AI engineers have explored distributed training in pursuit of lower time costs. Various techniques were proposed to accelerate distributed training and they will be discussed below.

### 2.1 Data Parallelism

Data parallelism is the most common parallelism technique due to its simplicity. In data parallel training, the model is replicated across the devices, and the dataset is split into several shards. Each dataset shard is fed to the model on one device as shown in Figure 3a. Collective communication is required to synchronize the parameter gradients after backward propagation [22]. Data parallelism makes it easy to train a model on multiple devices and scales sub-linearly with the number of devices.

One problem of data parallelism is that each device holds a copy of the model parameters, optimizer states, and gradients, leading to memory redundancy. When using stateful optimizers such as Adam [17], the optimizer states (i.e. momentum and variance) can occupy three times larger memory space compared to that occupied by the model parameters [12, 17]. To eliminate such redundancy, Zero Redundancy Optimizer was proposed in DeepSpeed [29] to partition these redundant model data over different devices during

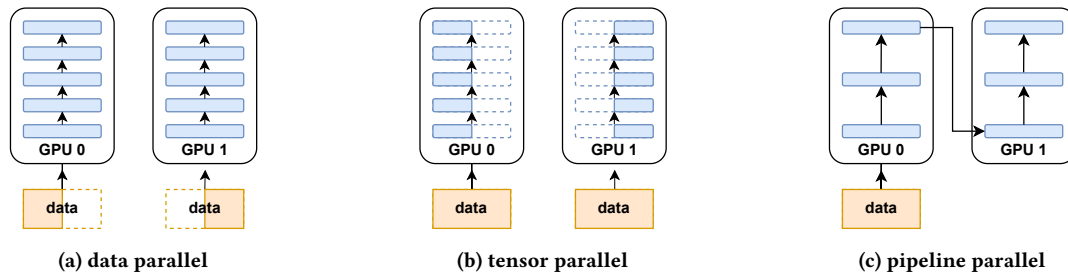


Figure 3: Existing parallelism for distributed training

data parallel training. As each device only holds a partition of gradients, optimizer states, and parameters, it will only update the partitioned parameters instead of the full model parameters on one device.

## 2.2 Model Parallelism

To go beyond data parallel training, more techniques were explored to shard the model parameters over a larger number of devices. As a result, model parallelism was proposed to tackle this problem. There are generally two types of model parallelism: tensor parallelism and pipeline parallelism.

### 1) Tensor Parallelism

Tensor parallelism shards the tensor over an array of devices and requires a distributed matrix-matrix multiplication algorithm for arithmetic computation as shown in Figure 3b. Megatron-LM [32] proposed 1D tensor parallelism which splits the linear layer in the row or column dimensions for the Transformer architecture [35]. More advanced tensor sharding mechanisms such as 2D [39], 2.5D [36], and 3D [4] were proposed to shard tensors in more dimensions. Collective communication is required among devices to ensure arithmetic correctness.

In Megatron-LM, the tensors are sharded in one dimension. Taking the Feed Forward module of the Transformer layer as an example, we can view the module as a matrix multiplication of  $Y = W_2 W_1 X$  as shown in Figure 4, where  $X$  is the input,  $W_1$  and  $W_2$  are the model parameters.  $W_1$  and  $W_2$  can be sharded vertically and horizontally respectively and produce a partial result of  $Y$  on each device. An all-reduce operation can be applied to the partial result to obtain the correct final result of the matrix multiplication. In this way, each device will only hold  $1/N$  of the parameters when training on  $N$  devices. This allows the model size to scale beyond the memory capacity of a single device.

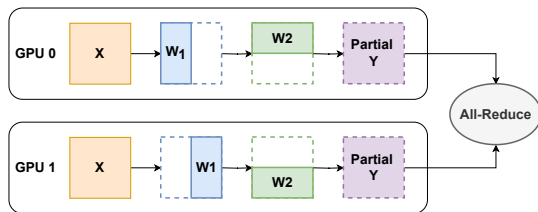


Figure 4: Megatron-LM MLP Module

One of the major problems of the 1D method is that it assumes the interconnect of devices has the same bandwidth. This makes it friendly only on machines with fully connected NVLinks among the GPUs on a single node as shown in Figure 9a. However, such high-end hardware is expensive and scarce. Many machines, even some in the supercomputing centers, only have partially connected GPUs as shown in Figure 9b. With this kind of GPU topology, the communication bandwidth between distant devices via the PCIe bus is much lower than that of directly connected GPUs. Therefore, the low communication bandwidth can hinder the efficiency of all-reduce operations in 1D tensor parallelism.

In addition, the 1D tensor parallelism has redundant memory usage in layer inputs and outputs. Taking the Feed Forward layer in the Transformer architecture as an example, the input  $X$  and output  $Y$  of the MLP layer are duplicated across different devices as shown in Figure 4. Such memory redundancy limits the maximum model size which can be trained on limited hardware resources, and is not helpful with the democratization of large-scale distributed training.

Besides 1D tensor parallelism, more advanced tensor parallelism is introduced for large-scale model training, namely 2D, 2.5D, and 3D tensor parallelism [4, 36, 39]. These methods split input, weight, and output tensors and thus have advantages in memory and communication efficiency, better coping with different hardware specifications. This provides the user with an option of using the most suitable tensor parallelism technique for their machines.

2D tensor parallelism [39] relies on the SUMMA and Cannon matrix multiplication algorithm [3, 6, 34] and splits a tensor along two different dimensions. Given  $N$  devices arranged in a square network topology, a tensor of shape  $[P, Q]$  will be partitioned into a chunk tensor of shape  $[P/\sqrt{N}, Q/\sqrt{N}]$ . 2.5D Tensor Parallelism [36] was inspired by 2.5D matrix multiplication algorithm [33] and proposed to further parallelize 2D tensor parallelism. It adds the optional *depth* dimension of the matrix for parallelization. When *depth* = 1, it is close to 2D tensor parallelism. When *depth* > 1, it partitions the matrix 3 times and adds one more degree of parallelization. Given  $N$  devices, the tensor is split in a way such as  $N = S^2 * D$  where  $S$  is the size of one side of the square and  $D$  is the depth of the cuboid. 3D tensor parallelism [4] was proposed based on the 3D matrix multiplication algorithm [1]. 3D tensor parallelism splits a tensor in a cubic manner. As not every tensor has 3 dimensions, we choose to partition the first and last dimension only where the first dimension will be partitioned twice. For example, a tensor

of shape  $[P, Q]$  will be partitioned into a chunk tensor of shape  $[P/\sqrt[3]{N^2}, Q/\sqrt[3]{N}]$ .

As the advanced tensor parallelism methods require different network topologies, the user needs to choose the method based on the number of GPUs. 1D method can work with any number of GPUs while 2D, 2.5D and 3D methods require the  $n^2$ ,  $a * n^2$ , and  $n^3$  GPUs respectively, where  $a$  and  $n$  are positive integers. The user can fall back to 1D tensor parallelism when the number of GPUs does not fulfill the requirement. These advanced tensor parallelism methods provide lower communication volume when scaling to a larger number of devices [1, 3, 6, 33] and this will be further discussed in Section 3.1.

#### 2) Pipeline Parallelism

Methods such as PipeDream [25], GPipe [16], and Chimera [20] were proposed to split the model into several chunks of consecutive layers and each chunk is allocated to a device as shown in Figure 3c. Intermediate activations and gradients are passed between pipeline stages to complete the forward and backward pass. As a result, this method reduces cross-node communication. Pipeline parallelism allows multiple devices to compute simultaneously, leading to a higher throughput. One drawback of pipeline parallel training is that there will be some bubble time, where some devices are idle when others are engaged in computation, leading to the waste of computational resources [25].

### 2.3 Sequence Parallelism

Tensor parallelism mainly tackles the memory bottleneck brought by model data. However, the non-model data can be the bottleneck in applications such as AlphaFold and document-level text understanding. This is because these applications rely on long-sequence data. As the self-attention module in the Transformer layer is of quadratic complexity with respect to the sequence length, long-sequence data will increase the memory usage consumed by the intermediate activation, limiting the training capability of the devices.

Sequence parallelism [21] is proposed to enable long-sequence modeling by breaking the memory wall brought by the large sequence dimension. In sequence parallelism, the model is replicated across devices just like data parallelism. The input data is split along the sequence dimension and each device only keeps a sub-sequence. The self-attention module is replaced with the Ring Self-Attention module such that the partial query, key, and value embeddings are exchanged among devices to complete the self-attention calculation.

### 2.4 Heterogeneous Training

To further expand the memory capacity of a single device, DeepSpeed proposed zero-offload [30] which moves the tensors from GPU to CPU or NVMe disks when not in use to make room for larger models. It is often seen that CPU memory is much larger than the GPU memory on machines such as the Nvidia DGX1 workstation. By utilizing high-performance heterogeneous storage devices and appropriately swapping tensors between different hardware devices, it became possible to train a model with billions of parameters on a single GPU. This is especially friendly to users with

limited computing resources and essential for the democratization of large-scale model training.

### 2.5 Automatic Parallelization

The latest advance in parallel training is the automatic selection and execution of parallelization strategies as demonstrated in FlexFlow [23] and Alpa [42]. Alpa was proposed recently to automatically search for a suitable parallelization plan including data and model parallelism given the cluster mesh. It then compiles the computation graph into distributed sharded graph with communication operators and runs the compiled executable on the cluster. However, Alpa is not made to be hardware-aware and does not automatically consider the network topology. Meanwhile, it does not search for other optimization techniques such as activation checkpointing, leading to suboptimal results.

## 3 DESIGN

Colossal-AI is featured by an array of acceleration techniques constructed in a modular way, which can cover a wide range of training settings to achieve maximal performance. It addresses the difficulties in achieving consistent acceleration in deep learning training due to diverse hardware conditions, met by Megatron-LM and DeepSpeed as well. This section will discuss the implementation and analysis of the acceleration techniques integrated in Colossal-AI.

### 3.1 Multi-dimensional model parallelism

First of all, Colossal-AI provides an array of model parallelism methods to cater to the needs of distributed training. Thus, it allows the model size to scale to billions of parameters by sharding the model over devices. In Colossal-AI, all existing tensor parallelism methods are supported so that the user can choose one method based on their training requirements and the number of GPUs while Megatron-LM only supports 1D tensor splitting. As tensor parallelism is mainly applied to matrix-matrix multiplication, it is highly suitable for the acceleration of Transformer models which widely uses linear layers.

Among all tensor parallelism methods, one prominent advantage of advanced tensor parallelism, namely 2D, 2.5D, and 3D tensor parallelism, is that it has a lower communication cost compared to 1D tensor parallelism. Table 1 has shown the total communication volume when computing a matrix multiplication  $Y = WX$  where  $X$  is of shape  $(b, s, h)$ ,  $W$  is of shape  $(h, h)$  and  $Y$  is of shape  $(b, s, h)$ . In Table 1, the following notations are used.

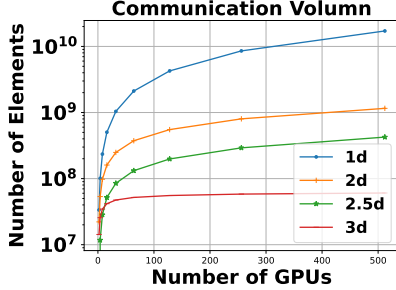
- $p$ : the total number of GPUs
- $j$ : the number of GPUs on one side of the square-shaped network topology, where  $p = j^2$
- $k$ : the number of GPUs on one side of the front square of the cuboid-shaped network topology, where  $p = d * k^2$
- $d$ : the number of GPUs in the depth dimension of the cuboid-shaped network topology
- $l$ : the number of GPUs on one side of the cube-shaped network topology, where  $p = l^3$
- $S_X$ : the number of elements in the input matrix  $X$ , the same semantic is applied to  $S_W$  and  $S_Y$ .
- $b$ : batch size of the input matrix  $X$ .
- $s$ : the sequence length of the input matrix  $X$ .

- $h$ : the hidden size of the weight  $W$ .

Mode	Total Communication Volume (number of elements transferred)
1D	$2(p-1) * S_x$
2D	$3(j-1) * (S_x + S_w)$
2.5D	$3(k-1) * (S_x/d + S_w)$
3D	$2(l-1)/l * (S_x + S_w + S_y)$

**Table 1: Communication Volume of Tensor Parallelism**

As shown in Figure 5, the communication volume of the advanced tensor parallelism is significantly lower than that of 1D tensor parallelism, especially when a large number of nodes is used. The underlying reason for communication efficiency is that advanced tensor parallelism only incurs communication on a subgroup of the computing nodes. For example, in 2D parallelism, collective communication only involves the nodes in one row or one column of the square-shaped network. In contrast, 1D tensor parallelism involves all computing nodes for one collective communication call. Therefore, advanced tensor parallelism allows scaling beyond one node while 1D tensor parallelism is often restricted to intra-node computing.



**Figure 5: Scaling Performance of Tensor Parallelism in Theoretic Analysis** ( $h = 1024, s = 512, b = 32$ )

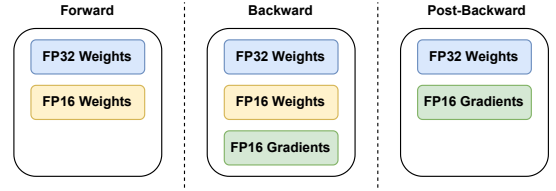
Besides tensor parallelism, Colossal-AI has also included sequence parallelism and pipeline parallelism so that hybrid parallelism is available out of the box to accelerate model training in large-scale clusters.

### 3.2 Enhanced Sharding and Offloading

Zero redundancy data parallel training and offloading proposed by DeepSpeed enable large-scale model training. However, it is still bound to the CPU-GPU and GPU-GPU communication and its rigid implementation leads to poor extensibility. Colossal-AI has re-designed the tensor sharding and offloading mechanism for better performance. Colossal-AI proposed a unified sharded tensor interface and supports customizable sharding strategies and life-cycle hooks for easy modification of the training workflow. As such, zero-redundancy data parallel can be easily supported and extended. Meanwhile, it also integrates the chunk strategy

proposed in PatrickStar [12] to arrange tensors in chunks to further improve the communication bandwidth utilization and memory usage, making tensor offloading more efficient.

Such flexible design brings several benefits. Firstly, it enables the re-use of FP16 storage space in the memory so that larger models can be trained. In the forward pass, we hold FP16 parameters. In the backward pass, when the gradients are computed, the FP16 parameters are no longer needed. We can thus save FP16 gradients in the same storage space which holds FP16 parameters during forward as shown in Figure 6. In this way, Colossal-AI further reduces redundancy and peak memory usage and the CPU memory can afford to accommodate larger models.



**Figure 6: Memory Space Reuse**

Secondly, an adaptive tensor placement and parameter update can be enabled during heterogeneous training. In DeepSpeed’s zero offloading, it provides an implementation of CPU Adam to update the model parameters in the CPU. However, this method requires all the FP32 master model weights to be placed in the CPU memory. In Colossal-AI, we implemented an adaptive hybrid Adam optimizer instead. During heterogeneous training, Colossal-AI’s hybrid Adam optimizer monitors the available memory space on the GPU. It does not statically keep all FP32 weights in the CPU memory, instead, it dynamically keeps part of parameters and gradients on the GPU as long as there is space left. In this way, parameters are updated on both CPU and GPU, leading to better resource utilization and lower communication cost.

### 3.3 Automatic Parallelization on Dynamic Computation Graph

Inspired by Alpha [42], Colossal-AI has included an experimental automatic parallelism feature to improve upon the Alpha project. One challenge in automatic parallelization is the sharded tensor conversion. For example, a tensor sharded on its 0th dimension can be converted to the one sharded in the last dimension. Alpha hardcodes a conversion table, but this limits the number of sharded dimensions to keep the table reasonably small. We implemented a greedy algorithm to search to speed up sharding conversion and increase the number of sharding dimensions. Moreover, we integrate activation checkpoint into the search problem such that a model can be both sharded and activation checkpointed to achieve maximum performance. As this feature is only experimental, it will be discussed separately in another paper as a future work.

## 4 IMPLEMENTATION

The overall architecture of Colossal-AI is shown in Figure 1. It has a parallel context manager that manages the meta information of the



complex hybrid parallel distributed environment and automatically switches to the corresponding parallel mode based on the parallel context. It has a user-friendly interface for building tensor-parallel models and various acceleration tools, including activation checkpointing and mixed precision training. It also has an execution engine and trainer that provide extensibility for user customization, allowing them to define their own training schedule and hooks at the operator or trainer level.

**4.0.1 Modularity.** The principle of modularity and extensibility is upheld throughout the development and the different acceleration techniques can easily be combined in pursuit of maximal performance.

**4.0.2 Extensibility.** As a system under constant development, Colossal-AI provides various interfaces to implement customized functions for future extensions. For example, the sharding module allows the user to define their own sharding strategy and life-cycle hooks in order in an attempt to explore more efficient training methods.

**4.0.3 User-Friendliness.** To minimize the change to the user code, Colossal-AI provides user-friendly APIs for model training. The user only needs to prepare a configuration that specifies the features by following a pre-defined schema. Colossal-AI will then inject the acceleration features into the execution engine with ‘colossalai.initialize’ as shown in Listing ??.

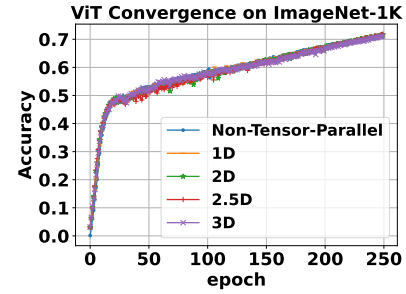
Colossal-AI also provides parallelized popular model components such as BERT [8], GPT [28], ViT [9], which the users can use directly. This does not require the users to have domain expertise so that they do not have to manually design their parallelism strategy like GShard [19].

```

1 import colossalai
2
3 # specify using 1D tensor parallelism with parallel size 4
4 config = dict(parallel=dict(
5     tensor=dict(
6         size=4,
7         mode='1d'
8     )
9 )
10 )
11
12 # launch distributed network
13 colossalai.launch_from_torch(config=config)
14
15 # define your training components
16 ...
17
18 # initialize with Colossal-AI
19 engine, trainloader, _ = \
20     colossalai.initialize(model,
21                           optimizer,
22                           criterion,
23                           trainloader)
24
25 # run training
26 for data, label in train_dataloader:
27     engine.zero_grad()
28     output = engine(data)
29     train_loss = engine.criterion(output, label)
30     engine.backward(train_loss)
31     engine.step()

```

**Listing 1: Colossal-AI Usage**



**Figure 7: Convergence Performance of ViT on ImageNet**

## 5 EVALUATION

### 5.1 Experiment Setup

To holistically evaluate the system performance of Colossal-AI, we have conducted various experiments on different hardware. The system specification is listed in Table 2. Due to resource constraints, we only tested a portion of the prominent features on each system as stated in the *Experiment Item* column. We used Megatron-LM and DeepSpeed as our baselines for experiments and Megatron-LM tensor parallelism is annotated as 1D tensor parallelism in the results.

### 5.2 Multi-Dimensional Tensor Parallelism

#### 1) Convergence

Experiments were conducted with Vision Transformer (ViT) [9] on the ImageNet-1k dataset to verify the arithmetic correctness and numerical stability of multi-dimensional tensor parallelism on System III. The ViT model has 12 Transformer layers with 384 hidden size and 6 attention heads. We used Jax initialization and AdamW optimizer with 0.003 learning rate and 0.3 weight decay. The input image is of shape 224 and the patch size is 16. The global batch size is 4k and the model is trained for 250 epochs. As shown in Figure 7, the testing accuracy curves of Multi-Dimensional tensor parallelism well align with that of the PyTorch data parallel training.

#### 2) Memory Efficiency

As 2D, 2.5D, 3D tensor parallelisms partition the input data, layer weight, and output activation while 1D tensor parallelism only partitions the layer weight, the former is expected to have lower memory consumption. As a result, the first three methods allow the GPUs to accommodate larger models. To demonstrate the memory efficiency, we have conducted two range tests which scale by batch size and hidden size on System I. In this range test, we created a model which consists of two linear layers. We run 1D, 2D and 2.5D experiments on 4 GPUs and 1D, 2.5D (depth=2), and 3D experiments on 8 GPUs. We measure the max allocated CUDA memory during the forward and backward pass, and the results are shown in Figure 8. The memory consumption of 1D tensor parallelism is much higher than those of 2D, 2.5D, and 3D tensor parallelism. With the batch size equal to 512 and 8 GPUs, the memory consumption of 2.5D and 3D is 44% and 65% lower than that of 1D tensor parallelism respectively in Figure 8b. With the hidden size of 16384 and 8 GPUs, the memory performance of

System ID	#GPUs per node	#Nodes	GPU Model	GPU Interconnect	Cross-node Interconnect	Experiment Item
I	8	1	Nvidia A100 (80GB)	NVLink	N/A	Tensor Parallelism
II	8	1	Nvidia A100 (80GB)	NVLink between adjacent GPUs, PCIe between distant GPUs	N/A	Tensor Parallelism, ZeRO
III	4	16	Nvidia A100 (40GB)	NVLink	InfiniBand HDR (200Gb/s), and Dragonfly network topology	Tensor Parallelism, Sequence Parallelism
IV	1	64	Nvidia P100 (16GB)	RDMA	Cray Aries routing and communications ASIC, and Dragonfly network topology	Tensor Parallelism

Table 2: System Specification for Experiments

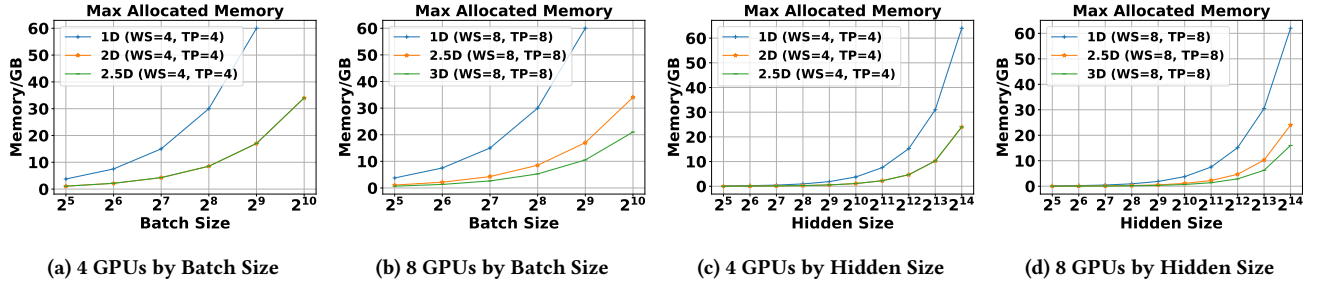


Figure 8: Range Test for Memory Consumption of Tensor Parallelism with 4/8 GPUs

2.5D and 3D tensor parallelism is 62% and 74.2% better than that of 1D tensor parallelism respectively in Figure 8d. Therefore, more advanced tensor parallelism is a better option when scaling to super large-scale models.

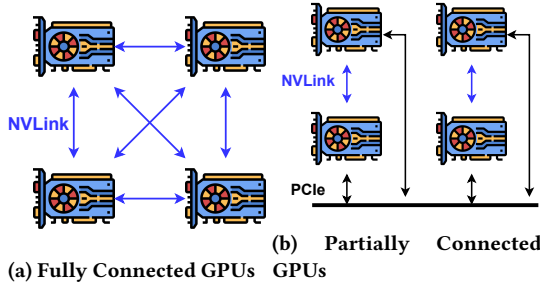


Figure 9: Common network topology on GPU nodes

### 3) Hardware Compatibility

Experiments were conducted on Systems I and II to further investigate the impact of GPU interconnect on the performance of tensor parallelism. System I and System II were selected for experiments as the former have fully connected NVLink between any pair of GPUs as shown in Figure 9a while the latter only has NVLink between 4 pairs of adjacent GPUs as shown in Figure 9b. The communication bandwidth of System I is consistently high regardless of whether it is measured for a pair of GPUs or a group of GPUs as shown in Figure 10. However, the communication bandwidth drops significantly from 184 GB/s to 15 GB/s when the communication occurs among

non-adjacent GPUs as only adjacent GPUs have high-performance NVLink.

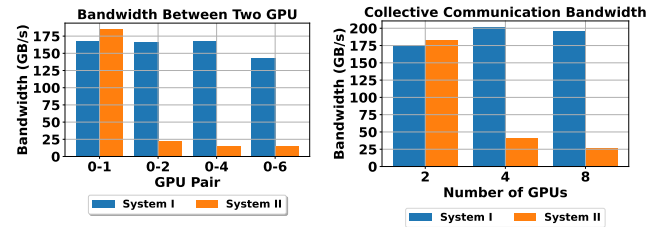


Figure 10: Communication Bandwidth on System I and II (broadcasting 125 MB data using the NCCL Bandwidth Test tool)

The GPU topology of System II is therefore not friendly to 1D tensor parallelism which relies on all-reduce operations across all the GPUs via PCIe. Instead, the 2D and 2.5D only have communication between a pair of GPUs instead of across all GPUs. This allows part of the communication to still utilize the high NVLink bandwidth between adjacent GPUs.

We trained ViT on the ImageNet-1k dataset with different configurations for 4 GPUs and 8 GPUs on both System I and II. On 4 GPUs, the ViT model has 64 Transformer layers with hidden size of 3072 and 48 attention heads. On 8 GPUs, the hidden size and the number of attention heads are adjusted to 4096 and 64 respectively as there

#GPUs	Mode	#Transformer Layer	Hidden Size	#Attention Heads	Batch Size	Throughput (img/sec)	Speedup over 1D (%)
4	1D	24	2048	32	128	5.06	-
	2D				256	6.18	22.1
	2.5D				256	6.73	33.0
8	1D	24	2048	32	256	7.46	-
	2.5D				384	6.57	-11.9
	3D				512	8.38	12.3
16	1D	32	4096	64	64	3.42	-
	2D				256	5.33	55.8
	2.5D				256	5.46	59.6
32	1D	32	4096	64	128	4.22	-
	2.5D				256	5.46	50.6
64	1D	32	4096	64	128	4.63	-
	2D				512	12.76	275.5
	2.5D				512	4.93	6.5
	3D				512	8.63	86.4

Table 3: Performance of Tensor Parallelism with Different Number of GPUs

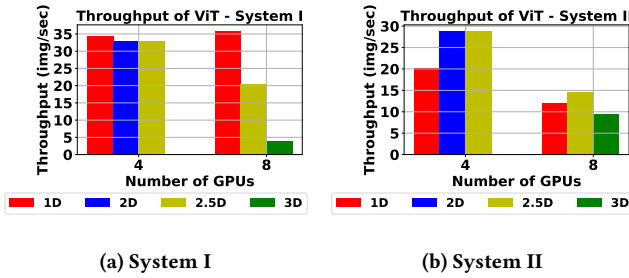


Figure 11: Throughput of ViT Training on System I and II

is more memory available. The model is trained with increasing batch size until the out-of-memory problem occurs. As such, we present the best throughput for each tensor parallelism method. In Figure 11a, the throughput of 2D, 2.5D, and 3D tensor parallelism cannot compete with 1D tensor parallelism on both 4 GPUs and 8 GPUs. This is expected for two reasons. The first reason is that 1D tensor parallelism can utilize the high communication bandwidth with all GPUs involved in System I. The second reason is that 2D, 2.5D, and 3D tensor parallelisms have more communication volume with a small number of processors and will only surpass 1D tensor parallelism when the number of processors increases.

However, when the experiment is switched to System II in Figure 11b, 1D tensor parallelism will encounter a bottleneck due to the low communication bandwidth in collective communication across 4 and 8 GPUs. Meanwhile, 2D and 2.5D can deliver a throughput that is 40% higher than that of 1D tensor parallelism with 4 GPUs. With 8 GPUs, 2.5D tensor parallelism can still outperform 1D tensor parallelism by 20.6%. 3D tensor parallelism still delivers lower performance than 1D tensor parallelism due to the low scaling.

#### 4) Throughput Comparison

To test the performance of tensor parallelism with more GPUs, we trained ViT on System IV. As System IV only has 16 GB GPU memory, therefore, we adjusted the configuration of the ViT model accordingly. The model is set to have 24 layers with the hidden size

of 2048 and 32 attention heads for 4 and 8 GPUs. From 16 GPUs onwards, the model is set to have 32 layers with the hidden size of 4096 and 64 attention heads.

The results for 4 to 64 GPUs are shown in Table 3. It can be observed that as the number of GPUs increases, the speedup of advanced tensor parallelism over 1D tensor parallelism increases up to 2.76. This can be attributed to the lower communication volume of advanced tensor parallelism methods when scaling to more processors. Together with memory efficiency and low communication volume, 2D, 2.5D, and 3D tensor parallelism is a better option for large-scale distributed training.

### 5.3 Sequence Parallelism

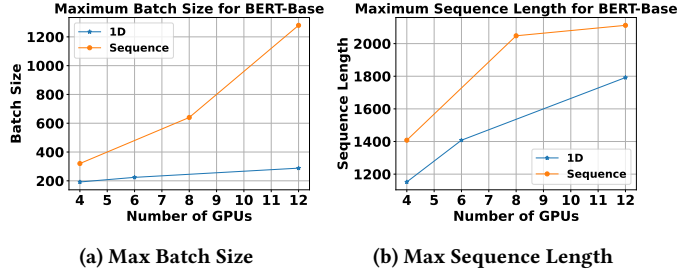
In this section, we compare Sequence Parallelism with 1D tensor parallelism for memory efficiency and training throughput. As Sequence Parallelism is designed for situations where activations consume more memory than model data, BERT-Base is chosen as our experiment model and trained on the Wikipedia dataset [13]. We conducted the experiments on System III. It should be noted that 1D tensor parallelism requires the number of attention heads (12) to be divisible by the parallel size, we can only use 4, 6, and 12 GPUs whereas the 6-GPU experiment uses 2 nodes and 3 GPUs from each node. Meanwhile, Sequence Parallelism is not limited by the number of attention heads, thus we conducted experiments on 4, 8, and 12 GPUs.

#### 1) Memory Efficiency

We increase the batch size and sequence length until the out-of-memory problem occurs for both 1D tensor parallelism and Sequence Parallelism. The sequence length is fixed at 512 for the batch size test while the batch size is fixed at 64 for the sequence length test.

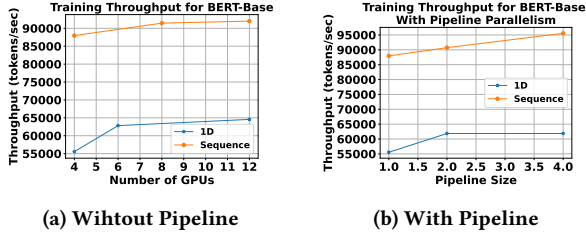
As shown in Figure 12a, Sequence Parallelism can reach larger batch size than 1D tensor parallelism. This is because that 1D tensor parallelism has a memory bottleneck in the duplicated activations where the activation is split along the sequence dimension in Sequence Parallelism. The maximum batch size of Sequence Parallelism is 4.44 times larger than that of the 1D tensor parallelism





**Figure 12: Memory Efficiency of Sequence Parallelism over 1D Tensor Parallelism**

with 12 GPUs. The same pattern is observed in the maximum sequence length test as shown in Figure 12b. The maximum sequence length of Sequence Parallelism is 1.18 times larger than that of 1D tensor parallelism. If linear-complexity attention modules [37, 40] is used instead of the quadratic-complexity self-attention in BERT, Sequence Parallelism can achieve linear scaling of maximum sequence length with the number of GPUs, better supporting document-level text understanding.



**Figure 13: Training Throughput of BERT-Base**

## 2) Throughput Comparison

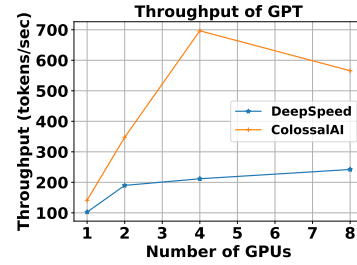
To evaluate the training speed of Sequence Parallelism, we trained BERT-Base with the sequence length of 512 and its maximum batch size. As shown in Figure 13a, Sequence Parallelism is up to 1.43 times faster than that of 1D tensor parallelism.

As sequence parallelism splits the input data and activation, it is naturally compatible with Pipeline Parallelism. While 1D tensor parallelism will split the activation before transferring the tensor to the next stage and gather it back afterward, Sequence Parallelism requires no such communication between pipeline stages. We further scaled the training with Pipeline Parallelism. The parallel size for both Sequence and 1D tensor parallelism is fixed at 4 and we scale the number of pipeline stages from 1 to 4. As shown in Figure 13b, Sequence Parallelism can train 1.55 times faster than 1D tensor parallelism with 4 pipeline stages.

## 5.4 Sharding and Offloading

In this section, we evaluated our own sharding and offloading module as discussed in Section 3.2 against DeepSpeed. We used DeepSpeed Stage 3 as the baseline, which partitions model parameters, gradients, and optimizer states in data parallel training. To demonstrate the capability of dynamic tensor placement in ColossalAI, we trained GPT-2 model with 10 billion parameters on the Wikipedia

dataset on System II. We set the batch size to 4 and scaled the data parallel training from 1 GPU to 8 GPU. As the batch size is small, the GPU memory is not completely used up. However, DeepSpeed’s static policy will still offload all model data to the CPU memory, leading to low memory efficiency and high communication overhead. Instead, Colossal-AI will dynamically determine whether a tensor should be placed on GPU or CPU depending on the memory availability. In this case, since Colossal-AI detects that there is still free memory on the GPU, it will only offload a small portion of the model data, leading to better utilization of the hardware resources and better training throughput as shown in Figure 14. We have also performed training on the OPT model [41] of 13 billion parameters with the batch size per GPU equal to 32. With a larger batch size, both systems utilized almost all GPU memory and Colossal-AI can still achieve 1.33 times speed up over DeepSpeed on 8 GPUs.



**Figure 14: Throughput of GPT Training with Sharding and Offloading with Batch Size 4**

## 6 FUTURE WORK

The Colossal-AI system is open-sourced and maintained on GitHub. One future work is to design a hardware-aware and efficient algorithm to automatically search for the optimal parallelization strategy as mentioned in Section 3.3. As an open-source project, we would actively integrate with model zoos such as Hugging Face Transformers [38]. We expect ColossalAI to be capable of parallelizing models in state-of-the-art model zoos so that distributed training can be more accessible to the Deep Learning community.

## 7 CONCLUSION

In this work, we designed and implemented Colossal-AI which integrated a vast number of advanced acceleration techniques into one unified system for large-scale distributed training. Colossal-AI comes with a flexible system design that supports an easy combination of different parallelism methods. In addition, its acceleration techniques provide robust performance under different hardware conditions and deliver superior performance compared to the baseline systems. In our experiments, we have demonstrated Colossal-AI can achieve up to 2.76x speedup over the baseline systems.

## REFERENCES

- [1] R. C. Agarwal, S. M. Balle, F. G. Gustavson, M. Joshi, and P. Palkar. 1995. A three-dimensional approach to parallel matrix multiplication. *IBM Journal of Research and Development* 39, 5 (1995), 575–582. <https://doi.org/10.1147/rd.395.0575>
- [2] Mandeep Baines, Shruti Bhosale, Vittorio Caggiano, Naman Goyal, Siddharth Goyal, Myle Ott, Benjamin Lefaudeux, Vitaliy Liptchinsky, Mike Rabbat, Sam

- Sheffer, Anjali Sridhar, and Min Xu. 2021. FairScale: A general purpose modular PyTorch library for high performance and large scale training. <https://github.com/facebookresearch/fairscale>.
- [3] Jarle Bernsten. 1989. Communication efficient matrix multiplication on hypercubes. *Parallel computing* 12, 3 (1989), 335–342.
  - [4] Zhengda Bian, Qifan Xu, Boxiang Wang, and Yang You. 2021. Maximizing Parallelism in Distributed Training for Huge Neural Networks. *arXiv preprint arXiv:2105.14450* (2021).
  - [5] Tom B Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. 2020. Language models are few-shot learners. *arXiv preprint arXiv:2005.14165* (2020).
  - [6] Lynn Elliot Cannon. 1969. *A cellular computer to implement the Kalman filter algorithm*. Montana State University.
  - [7] Tianqi Chen, Bing Xu, Chiyuan Zhang, and Carlos Guestrin. 2016. Training Deep Nets with Sublinear Memory Cost. <https://doi.org/10.48550/ARXIV.1604.06174>
  - [8] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805* (2018).
  - [9] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xi-aohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, et al. 2020. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929* (2020).
  - [10] Zhengxiao Du, Yujie Qian, Xiao Liu, Ming Ding, Jiezhong Qiu, Zhilin Yang, and Jie Tang. 2021. All NLP Tasks Are Generation Tasks: A General Pretraining Framework. *arXiv preprint arXiv:2103.10360* (2021).
  - [11] John Duchi, Elad Hazan, and Yoram Singer. 2011. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of machine learning research* 12, 7 (2011).
  - [12] Jiarui Fang, Yang Yu, Zilin Zhu, Shenggui Li, Yang You, and Jie Zhou. 2021. PatrickStar: Parallel Training of Pre-trained Models via Chunk-based Memory Management. <https://doi.org/10.48550/ARXIV.2108.05818>
  - [13] Wikimedia Foundation. [n. d.]. *Wikimedia Downloads*. <https://dumps.wikimedia.org>
  - [14] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep Residual Learning for Image Recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 770–778. <https://doi.org/10.1109/CVPR.2016.90>
  - [15] Yanping Huang, Youlong Cheng, Ankur Bapna, Orhan Firat, Dehao Chen, Mia Chen, HyukJoong Lee, Jiquan Ngiam, Quoc V Le, Yonghui Wu, and Zhifeng Chen. 2019. GPipe: Efficient Training of Giant Neural Networks using Pipeline Parallelism. In *Advances in Neural Information Processing Systems*, H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett (Eds.), Vol. 32. Curran Associates, Inc. <https://proceedings.neurips.cc/paper/2019/file/093f65e080a295f8076b1c5722a46aa2-Paper.pdf>
  - [16] Yanping Huang, Youlong Cheng, Ankur Bapna, Orhan Firat, Mia Xu Chen, Dehao Chen, HyukJoong Lee, Jiquan Ngiam, Quoc V Le, Yonghui Wu, and Zhifeng Chen. 2019. *GPipe: Efficient Training of Giant Neural Networks Using Pipeline Parallelism*. Curran Associates Inc., Red Hook, NY, USA.
  - [17] Diederik Kingma and Jimmy Ba. 2014. Adam: A Method for Stochastic Optimization. *International Conference on Learning Representations* (2014).
  - [18] Diederik P Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980* (2014).
  - [19] Dmitry Lepikhin, HyukJoong Lee, Yuanzhong Xu, Dehao Chen, Orhan Firat, Yanping Huang, Maxim Krikun, Noam Shazeer, and Zhifeng Chen. 2021. [GS]hard: Scaling Giant Models with Conditional Computation and Automatic Sharding. In *International Conference on Learning Representations*. <https://openreview.net/forum?id=qrwe7XHTmYb>
  - [20] Shigang Li and Torsten Hoefer. 2021. Chimera: Efficiently Training Large-Scale Neural Networks with Bidirectional Pipelines. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis* (St. Louis, Missouri) (SC '21). Association for Computing Machinery, New York, NY, USA, Article 27, 14 pages. <https://doi.org/10.1145/3458817.3476145>
  - [21] Shenggui Li, Fuzhao Xue, Yongbin Li, and Yang You. 2021. Sequence Parallelism: Long Sequence Training from System Perspective. <https://doi.org/10.48550/ARXIV.2105.13120>
  - [22] Shen Li, Yanli Zhao, Rohan Varma, Omkar Salpekar, Pieter Noordhuis, Teng Li, Adam Paszke, Jeff Smith, Brian Vaughan, Pritam Damania, and Soumith Chintala. 2020. PyTorch Distributed: Experiences on Accelerating Data Parallel Training. *Proc. VLDB Endow.* 13, 12 (aug 2020), 3005–3018. <https://doi.org/10.14778/3415478.3415530>
  - [23] Wenyuan Lu, Guihai Yan, Jiajun Li, Shijun Gong, Yinhe Han, and Xiaowei Li. 2017. FlexFlow: A Flexible Dataflow Accelerator Architecture for Convolutional Neural Networks. In *2017 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, 553–564. <https://doi.org/10.1109/HPCA.2017.29>
  - [24] Deepak Narayanan, Aaron Harlap, Amar Phanishayee, Vivek Seshadri, Nikhil R. Devanur, Gregory R. Ganger, Phillip B. Gibbons, and Matei Zaharia. 2019. PipeDream: Generalized Pipeline Parallelism for DNN Training. In *Proceedings of the 27th ACM Symposium on Operating Systems Principles* (Huntsville, Ontario, Canada) (SOSP '19). Association for Computing Machinery, New York, NY, USA, 1–15. <https://doi.org/10.1145/3341301.3359646>
  - [25] Deepak Narayanan, Aaron Harlap, Amar Phanishayee, Vivek Seshadri, Nikhil R. Devanur, Gregory R. Ganger, Phillip B. Gibbons, and Matei Zaharia. 2019. PipeDream: Generalized Pipeline Parallelism for DNN Training. In *Proceedings of the 27th ACM Symposium on Operating Systems Principles* (Huntsville, Ontario, Canada) (SOSP '19). Association for Computing Machinery, New York, NY, USA, 1–15. <https://doi.org/10.1145/3341301.3359646>
  - [26] Deepak Narayanan, Mohammad Shoeybi, Jared Casper, Patrick LeGresley, Mostofa Patwary, Vijay Korthikanti, Dmitri Vainbrand, Prethvi Kashinkunti, Julie Bernauer, Bryan Catanzaro, Amar Phanishayee, and Matei Zaharia. 2021. Efficient Large-Scale Language Model Training on GPU Clusters Using Megatron-LM. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis* (St. Louis, Missouri) (SC '21). Association for Computing Machinery, New York, NY, USA, Article 58, 15 pages. <https://doi.org/10.1145/3458817.3476209>
  - [27] Denis Paperno, Germán Kruszewski, Angeliki Lazaridou, Quan Pham, Raffaella Bernardi, Sandro Pezzelle, Marco Baroni, Gemma Boleda, and Raquel Fernández. 2016. The LAMBADA dataset: Word prediction requiring a broad discourse context. 1525–1534. <https://doi.org/10.18653/v1/P16-1144>
  - [28] Alec Radford, Jeff Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. 2019. Language Models are Unsupervised Multitask Learners.
  - [29] Jeff Rasley, Samyam Rajbhandari, Olatunji Ruwase, and Yuxiong He. 2020. DeepSpeed: System optimizations enable training deep learning models with over 100 billion parameters. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 3505–3506.
  - [30] Jie Ren, Samyam Rajbhandari, Reza Yazdani Aminabadi, Olatunji Ruwase, Shuangyan Yang, Minjia Zhang, Dong Li, and Yuxiong He. 2021. ZeRO-Offload: Democratizing Billion-Scale Model Training. [arXiv:2101.06840](https://arxiv.org/abs/2101.06840) [cs.DC]
  - [31] Alexander Sergeev and Mike Del Balso. 2018. Horovod: fast and easy distributed deep learning in TensorFlow. *CoRR abs/1802.05799* (2018). [arXiv:1802.05799](https://arxiv.org/abs/1802.05799)
  - [32] Mohammad Shoeybi, Mostofa Patwary, Raul Puri, Patrick LeGresley, Jared Casper, and Bryan Catanzaro. 2019. Megatron-lm: Training multi-billion parameter language models using model parallelism. *arXiv preprint arXiv:1909.08053* (2019).
  - [33] Edgar Solomonik and James Demmel. 2011. Communication-Optimal Parallel 2.5D Matrix Multiplication and LU Factorization Algorithms. In *Euro-Par*.
  - [34] Robert A. van de Geijn and Jerrell Watts. 1995. *SUMMA: Scalable Universal Matrix Multiplication Algorithm*. Technical Report. USA.
  - [35] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is All you Need. In *Advances in Neural Information Processing Systems*, I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett (Eds.), Vol. 30. Curran Associates, Inc. <https://proceedings.neurips.cc/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf>
  - [36] Boxiang Wang, Qifan Xu, Zhengda Bian, and Yang You. 2021. 2.5-dimensional distributed model training. *arXiv preprint arXiv:2105.14500* (2021).
  - [37] Sinong Wang, Belinda Li, Madian Khabsa, Han Fang, and Hao Ma. 2020. Linformer: Self-Attention with Linear Complexity. *arXiv preprint arXiv:2006.04768* (2020).
  - [38] Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, Mariama Drame, Quentin Lhoest, and Alexander M. Rush. 2020. HuggingFace's Transformers: State-of-the-art Natural Language Processing. [arXiv:1910.03771](https://arxiv.org/abs/1910.03771) [cs.CL]
  - [39] Qifan Xu, Shenggui Li, Chaoyu Gong, and Yang You. 2021. An Efficient 2D Method for Training Super-Large Deep Learning Models. *arXiv preprint arXiv:2104.05343* (2021).
  - [40] Manzil Zaheer, Guru Guruganesh, Kumar Avinava Dubey, Joshua Ainslie, Chris Alberti, Santiago Ontanon, Philip Pham, Anirudh Ravula, Qifan Wang, Li Yang, and Amr Ahmed. 2020. Big Bird: Transformers for Longer Sequences. In *Advances in Neural Information Processing Systems*, H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin (Eds.), Vol. 33. Curran Associates, Inc., 17283–17297. <https://proceedings.neurips.cc/paper/2020/file/c8512d142ad849725f31a9a7a361ab9-Paper.pdf>
  - [41] Susan Zhang, Stephen Roller, Naman Goyal, Mikel Artetxe, Moya Chen, Shuohui Chen, Christopher Dewan, Mona Diab, Xian Li, Xi Victoria Lin, Todor Mihaylov, Myle Ott, Sam Shleifer, Kurt Shuster, Daniel Simig, Punit Singh Koura, Anjali Sridhar, Tianlu Wang, and Luke Zettlemoyer. 2022. OPT: Open Pre-trained Transformer Language Models. [arXiv:2205.01068](https://arxiv.org/abs/2205.01068) [cs.CL]
  - [42] Lianmin Zheng, Zhuohan Li, Hao Zhang, Yonghao Zhuang, Zhifeng Chen, Yanping Huang, Yida Wang, Yuanzhong Xu, Danyang Zhuo, Eric P. Xing, Joseph E. Gonzalez, and Ion Stoica. 2022. Alpa: Automating Inter- and Intra-Operator Parallelism for Distributed Deep Learning. In *16th USENIX Symposium on Operating Systems Design and Implementation (OSDI 22)*. USENIX Association, Carlsbad, CA, 559–578. <https://www.usenix.org/conference/osdi22/presentation/zheng-lianmin>