Scaling Llama 3 Training with Efficient Parallelism Strategies

Weiwei Chu, Xinfeng Xie, Jiecao Yu, Jie Wang, Amar Phanishayee, Chunqiang Tang, Yuchen Hao, Jianyu Huang, Mustafa Ozdal, Jun Wang, Vedanuj Goswami, Naman Goyal, Abhishek Kadian, Andrew Gu, Chris Cai, Feng Tian, Xiaodong Wang, Min Si, Pavan Balaji, Ching-Hsiang Chu, and Jongsoo Park

Meta Platforms, Inc.

Abstract

Llama is arguably the most widely used open-source large language model. This paper presents the design and implementation of the parallelism techniques used in Llama 3 pre-training. To operate efficiently on tens of thousands of GPUs, it employs a combination of four-dimensional (4D) parallelism: fully sharded data parallel (FSDP), tensor parallel (TP), pipeline parallel (PP), and context parallel (CP). Besides ensuring efficiency with parallelism choices and model co-design, we propose methods that also address other equally critical aspects: flexibility (e.g., novel PP supporting evolving batch sizes and heterogeneous model architecture, creative CP facilitating the introduction of model innovations like document-mask attention) and practicality (e.g., debugging performance and numerical issues at large scale). Furthermore, based on our production experience with large-scale Llama 3 pre-training, we provide several recommendations for future hardware design.

Keywords

Large Language Models, Training, Parallelism, Distributed Systems

1 Introduction

Large language models (LLMs) have revolutionized the field of natural language processing (NLP) by demonstrating remarkable capabilities across a diverse array of tasks, such as conversational agents, language translation, and code generation [3, 18, 28]. Beyond text-based applications, multimodal models extend the capabilities of LLMs by incorporating the ability to understand and generate content across multiple modalities, including audio, images, and video [2, 4, 19, 36, 40].

Llama [10, 37, 38] is arguably the most widely used open-source LLM, significantly impacting both industry and research communities. Llama 3, released on April 18, 2024 [23], features its largest model with 405B parameters, pre-trained on 16,384 H100 GPUs over several months, utilizing a total of 3.8×10^{25} FLOPs [10]. Training at this large scale necessitates a training framework with efficient parallelism strategies that shard the model and schedules the compute and communication across GPUs. As a result, Llama 3 was trained using a combination of four-dimensional (4D) parallelism techniques: fully sharded data parallel (FSDP) [30, 31, 44], tensor parallel (TP) [14, 15, 33], pipeline parallel (PP) [11, 16, 25], and context parallel (CP) [20].

Each of the above-mentioned parallelism techniques has its own performance trade-offs. When combined together, they compose a vast design space that needs to be carefully explored to achieve optimal performance. Many previous works have attempted to tackle this challenge from different angles [9, 12, 25, 29, 35, 36, 45]. Different from all these prior research studies, our system enables multi-dimensional parallelism, including context parallelism, for training the Llama 3 herd of models at an industrial scale of 16K GPUs [10]. While efficiency is undoubtedly important, from operational experience of training at scale we consider other dimensions, such as flexibility for dynamically changing configurations and practicality for debugging performance and numerics, as equally important.

Efficiency: Llama 3 pre-training is a capability computing problem where we maximize 16K GPUs to shorten the total pre-training time. It brings a unique challenge of limited parallelism degree across the data batch dimension. This limited batch size pushes larger model parallelism dimensions (TP and CP), and risks harming training efficiency due to the extra exposed communications and lower compute efficiency with smaller matrix multiplication (GEMM) shapes due to larger TP and CP sizes.

Flexibility: Llama 3 pre-training consists of multiple phases, each targeting different training goals (e.g., short context, long context, multimodal). The pre-training process configures these phases with different hyper-parameters (e.g., global batch size, sequence length), and heterogeneous model architectures (e.g., alternating self attention and cross attention layers, image encoder in multimodal), and resources (e.g., number of GPUs). Moreover, the document mask [10] brings an input-dependent attention mask that varies computation patterns across training batches. This dynamism in training workloads requires the system to be flexible enough to adapt to these changes while maintaining high efficiency.

Practicality: Enabling and optimizing combined parallelism at scale brings practical challenges, especially debugging at scale. Performance debugging to identify the root cause of a slowdown is challenging because the issue propagates through the whole system where the first observed problematic rank is not necessarily the source rank causing the problem. Moreover, debugging numerical discrepancies is challenging; e.g., it is hard to pinpoint a numerical gap in loss curves to either an implementation bug or a low-precision floating-point accumulation across parallel ranks.

In this paper, we detail the training framework for Llama 3, which builds upon past works while introducing new features to address the unique challenges of efficiency, flexibility, and practicality at scale. We summarize our contributions below:

- We enable 4D parallelism and identify optimal parallelism configurations to address batch size constraint and fit model into memory at 16K scale. Moreover, we co-design parallelism with model hyper-parameters (number of layers, layer types) to achieve both computation and memory balance for PP. When training the 405B model of Llama 3 on 16K GPUs, we achieve 400 TFLOPs per GPU with an 8K sequence length, and 380 TFLOPs per GPU with an 131K sequence length.
- We introduce a new pipeline parallel schedule that supports flexible global batch sizes and heterogeneous layer sharding. We demonstrate its applicability to Llama 3 multimodal pre-training. Furthermore, it strikes a balance between memory usage and

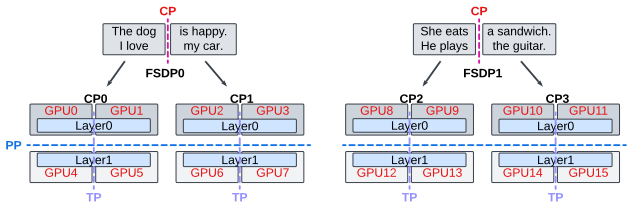


Figure 1: A two-layer LLM is sharded across 16 GPUs using 4D parallelism. FSDP and CP shard input data, with FSDP sharding along the batch size dimension and CP sharding along the sequence dimension. TP and PP shard model parameters, with TP sharding within the same layer and PP sharding across layers.

throughput. Moreover, we propose a novel all-gather based CP solution that allows for the easy introduction of model innovations (e.g., document-mask used in attention [10]) while achieving performance comparable to state-of-the-art baselines [21]. It also demonstrates strong scalability, specifically achieving a 3.89× latency reduction of attention layers on 4 GPUs compared to the single-GPU baseline.

- We share our methodology and the lessons of debugging performance and numerical issues at scale. This includes a top-down approach to analyze stacked performance traces to identify slowest rank of each parallelism dimension and trace-down across parallelism levels to identify the root-cause slow rank. We also share our method of debugging numerical gaps by splitting non-parallel implementations to rule out software bugs and identify key parts of the system where we need high-precision floating-point accumulations.

The rest of this paper is organized as follows: Section 2 provides background for parallelism and Llama 3 pre-training. Section 3 and Section 4 detail our novel PP and CP solutions respectively. Building the training system at scale, Section 5 and Section 6 share the key configurations of combined parallelism and the debugging methodology. Section 7 presents the system evaluation results, and Section 8 offers several recommendations for future hardware design drawing from our experience with Llama 3 pre-training before concluding in Section 9.

2 Background

In this section, we discuss the parallelism strategies adopted in Llama 3 pre-training. Then, we give an overview of the Llama 3 pre-training phases, including text and multimodal pre-training.

2.1 4D Parallelism

The ever-increasing LLM sizes make distribute training a must and we adopt 4D parallelism for distributed training.

Fully sharded data parallelism (FSDP): Conventional data parallelism, i.e., distributed data parallelism (DDP) [17], duplicates full model weights and distributes data batches across workers (GPUs). Workers need to synchronize their gradients globally at the end of each training iteration. Llama 3 pre-training adopts an in-house implementation based on Pytorch's fully sharded data parallelism (FSDP) [44] which supports further sharding the model weights and the related training parameters (gradients and optimizer states) across workers. In the rest of the paper, we use DP and FSDP interchangeably for simplicity. Our FSDP implementation supports three sharding strategies, following the same definition of Zero Redundancy Optimizer (ZeRO) from DeepSpeed [30] (ZeRO-1, ZeRO-2, and ZeRO-3), where model parameters, gradients, and optimizer states can be optionally sharded.

Tensor parallelism (TP): TP partitions the linear modules of the transformer layer across GPUs. Our tensor parallelism implementation follows the same idea introduced in Megatron-LM [33], which splits the GEMM operators along input or output dimensions. TP distributes the compute and memory costs across GPUs while introducing additional communication overheads. Sequence parallelism (SP) is often used in conjunction with TP to further reduce activation memory costs [14]. SP shards the sequences between the TP regions, and the sharded sequences are all-gathered and reduce-scattered before entering and exiting the TP region, respectively. It reduces memory costs while incurring additional communication overheads.

Pipeline parallelism (PP): PP splits model layers into several stages, and distributes the model stages onto different PP ranks. These stages are then executed in a pipelined fashion. Figure 2 shows one example of the interleaved 1F1B PP schedule [33]. There are two model stages on each PP rank. We call them virtual stages since they are not consecutive model layers (e.g., rank 0 has layer 0 and 3). We have 6 micro-batches and split them into 2 rounds. In each round, each virtual stage runs nc consecutive micro-batches, where $nc = 3$ (PP depth) in this figure (we run micro-batch 0-2 or 3-5 in one shot). The activation memory usage on each PP rank depends on the number of warm-up micro-batches. In the interleaved 1F1B, we require the batch size to be a multiple of the number of PP ranks.

Context parallelism (CP): CP shards input sequences without requiring changes to modules in the transformer block that are invariant to the sequence dimension (e.g., feed-forward networks). However, the attention module requires the full sequence for computation, necessitating additional communication to reconstruct the sequence. This can be implemented in various ways. Prior work [21] passes the key/value tensors of local token chunks among adjacent GPU ranks using ring-style communication while overlapping attention computation with communication. In contrast, we propose a straightforward all-gather-based CP approach, where the communication latency (all-gather) is fully exposed. We detail this simple yet flexible and efficient approach in Section 4.

2.2 Llama 3 Pre-training Overview

Llama 3 pre-training [10] encompasses three major phases: short context pre-training, long context text pre-training, and multimodal pre-training. Throughout the pre-training process, we gradually increased the number of GPUs, global batch sizes, and sequence lengths. For multimodal pre-training, we augmented the text model with additional cross-attention layers and introduced a trainable image encoder. The cross-attention layers take the image encoder outputs and the text outputs from the previous transformer layer as inputs, capturing the interactions between images and text. During pre-training, the original text model layers (i.e., non-cross-attention

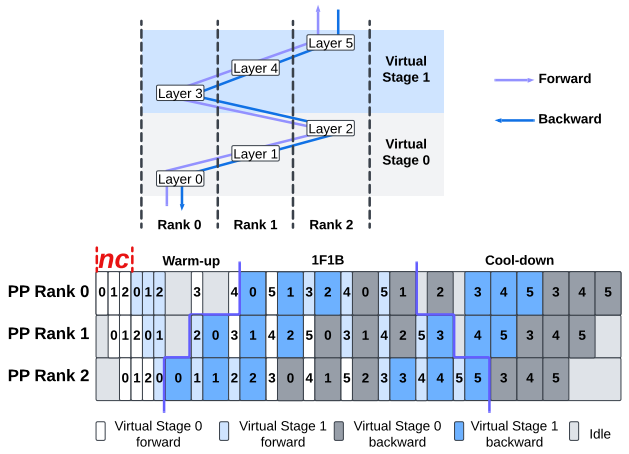


Figure 2: A 6-layer LLM is sharded across 3 PP ranks which executes 6 micro-batches using the 1F1B PP schedule [33]. Each rank holds two virtual stages, with each virtual stage containing one model layer. Model layers are placed in an interleaved manner, with layer 0 and layer 3 on rank 0, layer 1 and layer 4 on rank 1, and so forth. The 6 micro-batches are divided into two rounds, and each virtual stage processes nc consecutive micro-batches in each round, where $nc = 3$ in this example.

layers) are frozen, and only the cross-attention layers and image encoder are trained.

To optimize training efficiency, we employ 3D parallelism (FSDP, TP, and PP) for short context pre-training, and 4D parallelism (FSDP, TP, PP, and CP) for long context pre-training. For multimodal pre-training, we adopt a hybrid sharding method that shards the image encoder using 2D parallelism (FSDP, TP) and the text model using 3D parallelism.

In the following two sections, we present our novel PP and CP designs that achieve high efficiency and flexibility to support diverse training workloads. For the ease of illustration, we summarize all the parameters and their explanations used throughout the paper in Table 1.

3 Pipeline Parallelism

In this section, we present the design of PP and its application to multimodal training.

3.1 Design Overview

Our PP design is based on the interleaved 1F1B schedule [25]. We have made multiple optimizations and co-designed with Llama model during the pre-training to improve efficiency and flexibility. We illustrate our optimizations in detail as below.

3.1.1 Flexible PP schedule that supports arbitrary batch size. The *implementation* of the original interleaved 1F1B schedule constraints the batch size to be a multiple of the number of PP ranks [25]; while the idea supports an arbitrary number of micro-batches, changing the implementation is challenging. During Llama 3 training, the

Table 1: Parameters and definitions used in this paper.

Parameter	Definition
$ngpu$	Number of GPUs
seq	Sequence length
gbs	Global batch size
bs	Batch size per data parallel group
mbs	Micro-batch size in pipeline stage execution
$dp/tp/cp/pp$	GPU number in one data/tensor/context/pipeline parallel group
ndp	Number of data parallel group
v	Number of virtual stages on one PP rank
ppr	The index of PP ranks
nc	Number of continuous micro-batches for a virtual stage
nmb	Number of micro-batches for each virtual stage
tmb	Sum of nmb for v virtual stages on one PP rank

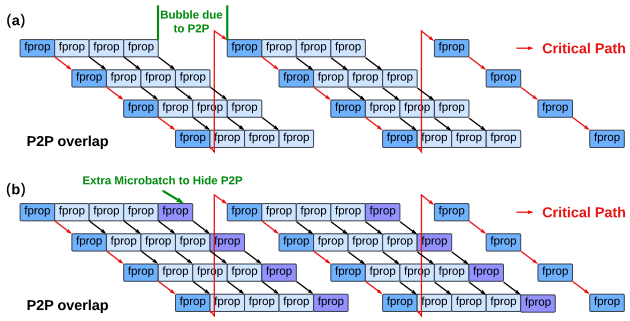


Figure 3: (a) In 1F1B schedule, bubble is introduced by exposed P2Ps (red arrows). (b) Running extra micro-batches (purple) helps reduce the bubble from exposed P2Ps.

global batch size is adjusted in multiple phases, which requires a PP schedule with flexible batch size support. We implement a flexible PP schedule that removes this constraint on number of micro-batches.

In the 1F1B schedule, the number of warm-up micro-batches from each pipeline stage is $(v-1)\times nc+2\times(pp-ppr\times v-1)$, where v is the number of virtual stages on each PP rank, nc is the number of consecutive micro-batches per stage, pp is the size of pipeline, and ppr is the index of pipeline rank. The total number of micro-batches on one PP rank tmb is the sum of micro-batches across all virtual stages $nmb \times v$, where nmb is the number of micro-batches. The original 1F1B schedule requires $nc == pp$ and $nmb\%nc == 0$. Note that in PP, there are certain phases when GPUs are idle waiting for the new micro-batches or tokens from the other PP ranks. We call these idle times as PP bubbles. In interleaved 1F1B, PP bubble ratio (defined as the PP idle time over the forward and backward compute time) is computed as $(pp-1)/nmb/v$ [33]. To reduce the PP bubble, we favor a smaller pp, more micro-batches nmb and more PP virtual stages v.

In our flexible PP schedule, nc can be set to any number between 1 and nmb, and nmb can be any number that is needed. When nc is greater than pp, we insert $nc - pp$ more micro-batches per virtual

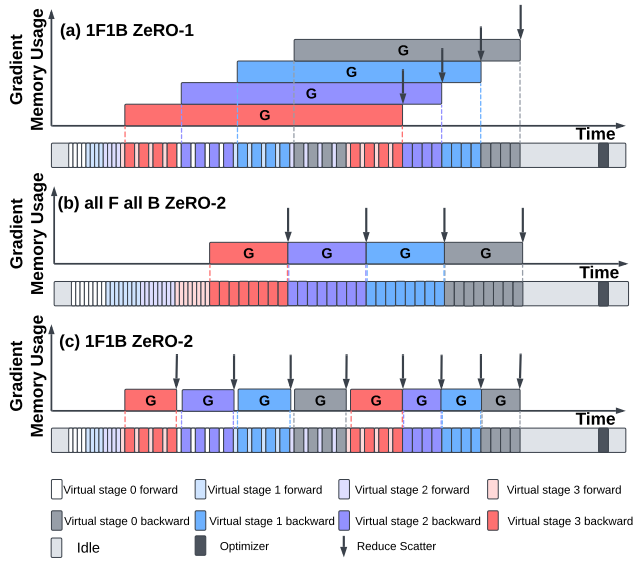


Figure 4: Gradient memory lifetime in different combinations of PP schedules and FSDP ZeRO modes: (a) 1F1B with ZeRO-1, reduce-scatter is launched only on the last micro-batch. (b) All forward all backward (same behavior between ZeRO-1/2. (c) 1F1B with ZeRO-2, reduce-scatter is launched on the last consecutive micro-batch.

stage into the warm-up phase. The extra micro-batches help hide the point-to-point communication as depicted in Figure 3, however, with a cost of increased peak memory usage due to $(nc - pp) \times (v - 1)$ more in-flight warm-up micro-batches than the original interleaved 1F1B schedule. When nc is smaller than pp, the schedule falls into the all forward and all backward schedule [11], where we execute the forward passes of all virtual stages before launching the backward, which is illustrated in Figure 4.

3.1.2 Balanced PP with model co-design. Sharding model layers uniformly to PP ranks would result in memory and computation imbalance. This is caused by having different number of warm-up micro-batches on different PP ranks and special model structures such as input embedding and output head which are placed only on the first and last PP ranks. As a result, training can run into out-of-memory (OOM) issues easily on the first PP rank with several GBs available on later ranks, or suffer from pipeline bubbles due to heavy last PP rank computation. To alleviate these issues, we co-designed the PP schedule with the model architecture by reducing one layer from the first pipeline rank to lower the peak memory across PP ranks, as well as one layer from the last pipeline rank to balance computation workload. With this change, the Llama 3 405B model is configured with 126 layers (instead of 128 layers at the very beginning).

3.1.3 Co-optimization of PP and FSDP. We consider FSDP ZeRO-1 and ZeRO-2 with PP. The 1F1B schedule alternates the execution of different virtual stages. As a result, gradients across executions of the same virtual stage need to be accumulated. FSDP ZeRO-2 with PP reshards gradients to save the memory, at a cost of more gradient reduce-scatters, as shown in Figure 4. In comparison, FSDP ZeRO-1 with PP keeps the unsharded gradients across the virtual stages, trade-offing the memory increase with less communication. In Llama 3 training, we choose FSDP ZeRO-1 with 1F1B schedule when $bs >= 2 \times pp$ and ZeRO-2 with all forward all backward when $bs < 2 \times pp$, to achieve a better performance. We find that with FSDP reduce-scatter at large scale the traffic congestion with other parallelisms can lead to degraded P2P performance.

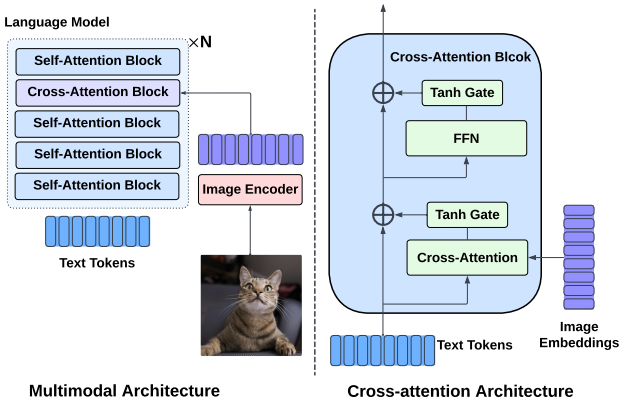


Figure 5: Illustration of Llama 3 multimodal architecture.

3.2 Case Study: Multimodal Training

In this section, we dive into the detail of Llama 3 multimodal training to show we adapt our PP to enable efficient and flexible training.

The Llama 3 multimodal model is composed with a pre-trained ViT image encoder [42] and the pre-trained Llama 3 text model. A set of transformer layers which use cross-attention (we call it cross-attention layers for the simplicity in the following text) are inserted into into every few original transformer layers from the text model that use the self-attention (we call it self-attention layers in the following text). The cross-attention takes inputs from both image encoder and self-attention layers. The model architecture is illustrated in Figure 5. During pre-training, self-attention layers are frozen, whereas the image encoder and cross-attention layers are trained. For training details, please refer to the Llama 3 technical report [24].

Due to the model architecture differences as mentioned above, we face two key challenges when scaling multimodoal pre-training.

- **Challenge 1: Sharding of image encoder**. We need to consider the sharding of one additional module, image encoder, whose compute and memory characteristics are different from the text model. Moreover, the sharding strategy must be general and flexible to scale well with different image encoder configurations; overfitting to one set of configurations can lead to poor performance on others.
- **Challenge 2: Workload imbalance of the text model**. The self-attention and cross-attention layers exhibit distinct compute and memory characteristics: (1) text sequence length is much shorter (less than 200 tokens) than the pre-trained text model (8K

tokens), and (2) the majority of the model weights (self-attention) is frozen. As a result, simply reusing the PP configuration from the text pre-training (which puts one transformer layer per virtual PP stage) leads to severe workload imbalance across PP ranks.

We illustrate how we adapt our PP designs to tackle these two challenges in the following sections.

3.2.1 Sharding of image encoder. We evaluated three candidate sharding choices before the implementation, as depicted in Figure 6.

Option 1: Shard the whole (image + text) model with PP. We put the image encoder on the first PP rank and run the image encoder together with the first text model virtual stage on the first PP rank for each micro-batch. Outputs from the image encoder are passed down along with transformer layer outputs to all other PP ranks through P2P communication.

Option 2: Separate the image and text model, and apply PP only to the text model. We separate image encoder from text model, run image encoder as a pre-processing stage of the text model on the first PP rank, and then broadcast image tokens to all pipeline stages and split into micro-batches to feed the text model which is trained with PP. After the text model pipeline finishes, the gradients of the image tokens are all-reduced. And we run the backward of the image encoder at last.

Option 3: Shard the image model across PP ranks. Shard or duplicate the image encoder across all PP ranks. Each image encoder replica will take a bs/pp portion of the input. The outputs are all-gathered across the PP stages and fed to the text model pipeline.

Among the three options, Option 1 requires minimal code changes as we could reuse the PP design for the text model, and only pack more tokens (to include the image tokens) for P2P communication. However, this design worsens the workload balancing issues for PP as the first PP rank is placed with more compute that includes the image encoder. As the configurations of image encoder could be changed during the training, this design is inflexible to adapt to the changes while maintaining high efficiency at the same time.

In comparison, Option 2 and 3 decouple the image encoder from the text model, and is more flexible to configure in the training. In our initial implementation, we adopted Option 2 for the ease of implementation. By putting the image encoder and further reducing the text model transformer layers on the first PP rank, we achieved a good training throughput. However, later during the training stages, the image resolution was significantly increased (from 448×448 to 672×672 pixels). And more transformer layers are added into the image encoder. Combining both factors, image encoder took a much longer latency (up to 33% of the combined image and text model training latency), leading to significant drop of the overall training throughput.

To address this issue, we switched to Option 3, where we replicate the image encoder on each PP rank, split the data batch into micro-batches and let the encoder compute them in parallel. This optimization helps reduce the encoder compute ratio from 33% to 8%, and recover the TFLOPs we achieved before the model changes.

3.2.2 Workload imbalance of text model. There are two key differences between the cross-attention and self-attention layers.

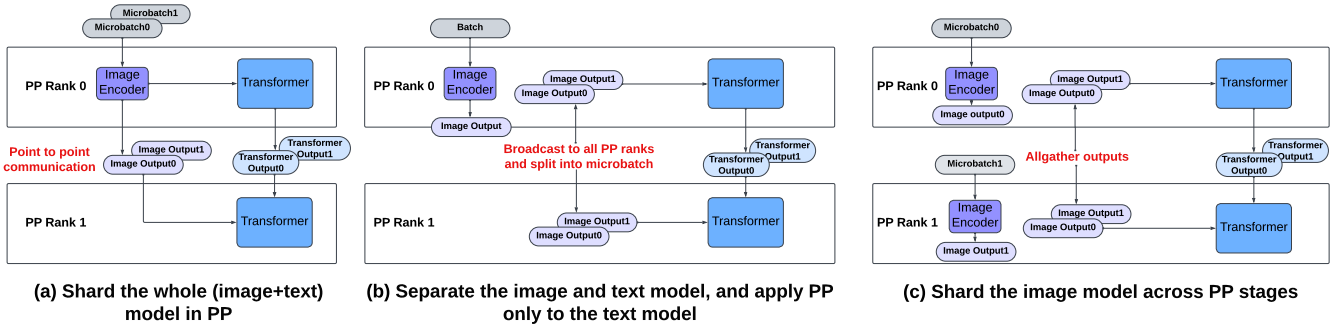
- Cross-attention takes both image sequence and text sequence as the inputs, whereas the self-attention only takes the text sequence at the input. Image sequence is much longer than the text sequence during pre-training (e.g., image sequence is 1.2K tokens for 448×448 resolution and 3K tokens for 672×672 resolution, while the text sequence is less than 200 tokens). As a result, the compute FLOPs of cross-attention layers in forward pass is much larger than the self-attention layers (depending on the ratio of image and text sequence lengths in the batch).
- Self-attention layers are frozen in the training. As a result, during the backward, self-attention layers only compute the input gradients, whereas the cross-attention layers compute both weight and input gradients, further exacerbating the workload imbalance between two layers.

When partitioning the text model for PP, we explored two options of placing self-attention and cross-attention layers: (1) wrap n self-attention layers and one cross-attention layer in one PP virtual stage; (2) wrap either n self-attention layers or one cross-attention layers in one virtual stage. Option 1 achieves a more balanced workload distribution across PP ranks, but leading to fewer PP virtual stages, and thus a larger PP bubble ratio. In comparison, Option 2 generates more PP virtual stages with a smaller PP bubble ratio. However, it is hard to achieve the workload balance given the workload difference as discussed above. In Llama 3 multimodal pre-training, we adopted Option 1 due to simplicity. We co-designed the multimodal model to decide on the final ratio of cross-attention layers and self attention layers (4:1) that achieves a reasonable training throughput, helping meet the production training deadline given the compute budget.

4 Context Parallelism

We enable context parallelism to support long context training of Llama 3 by splitting input tokens along the sequence length dimension. Although decreasing DP size dp to increase CP size cp needs to increase batch size per DP group bs correspondingly in order to keep the same global batch size gbs, PP in the combination of 4D parallelism makes the peak memory usage independent of bs. Thus when CP splits along the sequence length, it reduces the peak memory usage in spite of an increasing bs. More details about how each parallelism affects memory usage can be found in Section 5.

Design: In Llama 3, we propose and develop an all-gather based CP attention to deliver an efficient and flexible solution by all-gathering key(K)/value(V) tensors before attention computation. Although existing work, such as RingAttention [21], has solutions to overlap P2P communication of blocks of tokens with the computation, we still adopt all-gather based CP attention for two main reasons: First, Llama 3 model architecture requires the flexibility to support irregular attention masks, where existing work assumes the usage of full causal mask. In particular, the attention mask in Llama 3 enforces tokens to only attend other tokens from the same document (i.e., document mask), and this document boundary depends on the position of the end of sequence IDs (eos_ids) in input tokens. The computation of mask on every tile of tokens is error-prone, and irregular token communication makes it difficult to fully utilize network bandwidth. Second, all-gather approach



(a) Shard the whole (image+text) model in PP

(b) Separate the image and text model, and apply PP only to the text model

(c) Shard the image model across PP stages

Figure 6: Options to shard encoder with PP. (a) Place encoder on the first PP rank, communicate outputs from both image encoder and transformer layers across PP ranks. (b) Place encoder on the first PP rank, pre-process all inputs with image encoder and broadcast encoder output, then run the training of the text transformer model. (c) Replicate/shard encoder on all PP ranks, shard inputs across PP so that each encoder replica processes bs/pp portion of inputs. All-gather outputs before running the transformer.

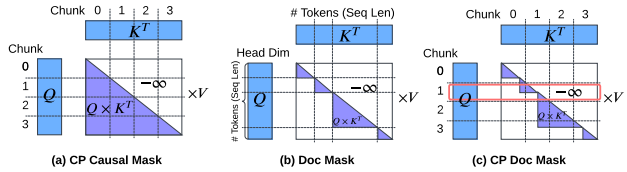


Figure 7: Examples of CP sharding for different masks.

does not bring much performance overhead in communication compared with ring-based approaches: Due to multi-group attention (MGA) or group query attention (GQA), the number of KV heads is smaller than the number of heads, thus both K and V tensors are smaller than Q tensor in attention. Moreover, the communication latency is under a time complexity of $O(seq)$ with the increase of sequence length (denoted as seq) while the time complexity of attention computation is $O(seq^2)$. This makes the exposed all-gather communication latency a smaller portion of CP attention when seq increases. With smaller sequence lengths (e.g., $seq = 8192$ or $seq = 16384$), all-gather based attention can still achieve a comparable performance to RingAttention because RingAttention needs to merge partial attention results with scaling and rescaling according to softmax log-sum-exp results [7, 8]. More concrete results about performance comparison between all-gather based and ring based CP attention can be found in Section 7.2.

Implementation: In CP attention implementation, we split input tokens evenly into $2 \times cp$ chunks, and each rank i processes both i-th and $(2 \times cp - i - 1)$-th chunks of tokens. This sharding and assignment can ensure the balance of computation workload among CP ranks. For example, Figure 7(a) shows a split of input tokens into 4 chunks for CP=2, and purple area stands for the computation workloads among token chunks with the causal mask (token i only attends token $0, 1, ..., i-1$). Also, Figure 7 shows an example of document mask in Llama 3 where tokens only attend other tokens from the same document. Although the document mask reduces the computation workloads, we still adopt the sharding strategy optimal for the full causal mask because the elapsed time of a train step is often bounded by the slowest rank of DP group and PP stages. In these cases, the slowest rank often processes the full long sequence without eos_id to split tokens.

The document mask particularly brings challenges to CP attention as the document boundary is irregular and input-dependent, and it does not align with the static CP sharding of tokens. In our all-gather based implementation, as we all-gather K and V tensors, computation on each Q token is independent: For example, Chunk 1 in Figure 7(c) only needs Chunk 0 and Chunk 1's K and V tensors to compute output results. But some tokens in Chunk 1 still need tokens from Chunk 0 as there are tokens from the same document across the CP sharding boundary. If we assume there are 16 tokens for examples in Figure 7 and the document length is [3, 3, 8, 2], the first two tokens in Chunk 1 need to attend all three tokens from the same document. To address this challenge, we pad Q sequence length with leading zeros. for attention computation outside this chunk of tokens, but retain K and V sequence length information as we have a full K and V tensors after all-gather. In this way, we can implement an efficient and accurate CP attention supporting the document mask.

Integration: When integrating CP into the end-to-end training system, we need to consider CP in following components: (1) Data parallelism group needs to consider both original DP and CP because CP splits input tokens along the sequence length but tokens processed by the same CP group still share the same set of model parameters. Thus it is effectively a part of DP when dealing with the communication of model parameters (all-gather parameters and reduce-scatter parameter gradients). (2) CP ranks in the same group select their local tokens, and use the whole sequence to compute its own attention mask: as detailed in our CP attention implementation, each CP rank needs a full sequence to compute KV seqlen and pad Q seqlen correctly. Thus each CP rank selects local tokens according to our sharding method where rank i takes both i-th and $(2 \times cp - i - 1)$-th chunks of tokens. We also need to select positional encodings [41] correspondingly. (3) Dataloaders still provide original DP groups with different batches of input training data.

Table 2: The size of each parallelism dimensions for Llama 3 pre-training of 405B models with 16M tokens per global batch on 16K GPUs (more details in the Table 4 of the Llama 3 technical report [10]).

Context Length	Global Batch Size	TP	CP	PP	DP
8,192	2,048	8	1	16	128
131,072	128	8	16	16	8

The split of sequence length is not visible to the tokenizer as each CP rank still needs the full sequence information for computing the attention mask.

5 4D Parallelism for Llama 3

We provide an overview of parallelism configurations for training Llama 3 at scale in Table 2. Training Llama 3 models on 16K GPUs is a capability computing problem where we aim to maximize the efficiency to shorten the total training time. At this scale, a fixed batch size of 16M tokens per training step limits the parallelism degree across training samples. In this section, we detail our reasoning process of finalizing parallelism configurations (i.e., the size of each parallelism dimension) to fully utilize 16K GPUs in training Llama 3 models with 16M tokens per global data batch.

5.1 The size of Parallelism Dimensions

We decouple the size of multiple parallelism dimensions into several key steps: First, we derive the minimal TP size given the limited global batch size. Second, we discuss the reason of using 3D parallelism rather than 2D parallelism in Llama 3. Finally, we detail the benefits of adding CP in long context training, and the configuration of our 4D parallelism. The definitions of symbols used in this section can be found in Table 1.

TP size: For a global token budget of 16M and sequence length seq 8K, global batch size gbs is 2048. Per-GPU batch size $bs = gbs/ndp$, where ndp is the number of data parallel groups and $ndp = ngpu/dp = ngpu/tp/pp/cp$. When using 2D parallelism, bs will be $gbs/ndp = 2K/(16K/tp/1/1) = tp/8$. In order to make $bs >= 1$, we need $tp >= 8$. When using 3D parallelism, bs will be $tp \times pp/8$, for efficient PP with less bubbles, we prefer $bs >= pp$, thus $tp >= 8$. Therefore, at the scale of training 16M tokens per step on 16K GPUs, we need $tp >= 8$ for both 2D and 3D. In our training cluster setting, each training node contains 8 GPUs. Configuring $tp <= 8$ can limit TP to use only intra-node communication (i.e., NVLink) which provides much higher bandwidth than inter-node network communication. In short, $tp = 8$ is the optimal size for TP under the batch size and hierarchical network bandwidth constraints.

2D or 3D parallelism: In order to fit model into memory, we consider either 2D parallelism (FSDP ZeRO-3 + TP) or 3D parallelism (FSDP ZeRO-1/2 + TP + PP). With the same TP size $tp = 8$ between 2D and 3D, the efficiency depends mainly on the overhead of FSDP and PP communications. For 2D with $bs = 1$, the computation latency is not long enough to hide FSDP communications while 3D has cheaper and more stable P2P communications, thus we chose 3D parallelism. For example, each model parameter contributes to 2 bytes communication data in FSDP ZeRO-3 (assuming BF16 data type) and 2 computation FLOPs for every token in forward. When training with 8K tokens ($bs = 1$ and $seq = 8192$), the arithmetic intensity of computation over communication is (2×8K) / 2 FLOPs per communication byte, which is much lower than the hardware ratio of peak computation FLOPs over network bandwidth, i.e. 989 TFLOPs for BF16 on Nvidia H100 GPU [26] over 50 GB/s RoCE network bandwidth (989K / 50 = 19.78K) [24].

When configuring 3D parallelism, we chose $pp = 16$ to fit model into memory. We did not consider FSDP ZeRO-3 with PP for three reasons: First, the model parallelism dimensions ($pp = 16$ and $tp = 8$) are large enough to accommodate 405B models. Second, FSDP ZeRO-3 has extra communication overheads on every PP stage forward and backward. Third, FSDP communications have performance interference when we overlap them with PP. In particular, inter-host P2P communications of PP are slowed down due to the resource contention of network hardware bandwidth between PP and FSDP communications.

4D parallelism for long context: For the long context phase of Llama 3 pre-training, the global batch size is reduced from 2K to 128 as the sequence length is increased to 128K while the number of tokens per global batch remains the same. Without changing the parallelism configuration, this immediately means bs drops to 1 which completely tanks performance due to unbearable bubble in PP. Decreasing DP leads to larger TP or PP in exchange for a larger bs. However, neither trade-offs is favorable because increasing PP at the same rate does not resolve the pipeline bubble issue, and increasing TP beyond 8 introduces expensive inter-host TP communications on the critical path. Given this, CP comes in as the perfect solution by sharding the sequence dimension within each training sample.

When we introduce CP to existing parallelism, we first need to consider replacing which parallelism dimensions into CP. When the global token 16M not changed but sequence length increased to 131K, gbs becomes 128. With batch size constraint $bs >= 1$, we can not replace TP or PP with CP, thus we can only replace DP with CP. With $tp = 8$ and $pp = 16$, As discussed previously $gbs/(ngpu/tp/pp/cp) >= pp$ is strongly preferred for PP efficiency, which means $cp >= 16$. We use $cp = 16$ to minimize CP communication overheads. Overall, CP allows us to easily scale to the long context training phase while keeping the same bs and pp configurations, and additionally reduces activation memory usage by sharding the sequence length dimension.

5.2 The Order of Parallelism Dimensions

We order parallelism levels to make the most efficient use of the network bandwidth. Our training cluster has a hierarchical network topology, ranging from high-bandwidth NVLink for GPUs within the same host as the innermost layer to lower-bandwidth cross-node networks as the outer layers. As a principle, we place the parallelism dimensions with higher communication demands (i.e. higher communication data volume, higher communication frequency, and/or communication latency more difficult to hide) into the inner levels of parallelism:

TP Communication: TP all-gathers and reduce-scatters activation or gradient tensors on every linear module. These communications

are fully exposed to the critical path, and happen four times in every transformer layer, two for the attention module and two for the feed-forward network (FFN) module.

CP Communication: CP all-gathers KV tensors or reduce-scatters the gradients of KV tensors on the inner-attention module. The communication latency is fully exposed and it happens once in each transformer layer. Despite having a similar communication data volume to PP, it involves *cp* ranks in a collective communication rather than PP's P2P between two ranks. Therefore, CP communication latency is longer as due to the synchronization among CP ranks.

PP Communication: PP communicates on every virtual pipeline stage. There is no synchronization between the two P2P ranks due to decoupled asynchronous P2P send and receive. When $pp = bs$, all P2P communications are fully exposed. Nevertheless, we prioritize CP into inner-layer over PP due to aforementioned reasons when analyzing CP communication.

DP Communication: DP with ZeRO-1/ZeRO-2 communicates only once per training step, i.e. all-gather model parameters and reduce-scatter gradients. Although its communication volume is comparable to PP, we can potentially hide its communication latency with forward / backward computation (i.e. all-gather parameters overlapped with model forward and reduce-scatter gradients overlapped with model backward). Thus we put DP as the outermost level in 4D parallelism.

In summary, we have a parallelism order of [TP, CP, PP, DP] from the innermost level to the outermost level considering communications along every parallelism dimension.

6 Debugging Parallelism at Scale

When enabling efficient multi-dimensional parallelism for training Llama 3 at scale, we need to debug performance, memory usage, and numerical issues of large scale jobs. We share our debugging process and lessons learnt from this process to facilitate future research and development.

6.1 Performance: Identifying Slow Rank

In multi-dimensional parallelism, debugging the performance at scale, especially identifying the root cause rank of the issue, is challenging due to the interactions of various parallelisms. These interactions complicate the process from observing a slow communication collective to root cause the rank causing the issue. Figure 8 shows an example of 8 GPUs with ($cp = 2$, $tp = 4$) configuration, and a stacked performance trace of TP communication collectives for 4 GPUs from a TP group. We can clearly observe that Rank 2 is the slowest rank of this group as its communication collectives are shortest, which indicates that TP collectives on other ranks are waiting for Rank 2 to join. However, this can hardly conclude Rank 2 is the bottleneck of the whole system as its slowness could come from its CP communication collectives where its peer rank (i.e., Rank 6) in the CP group is the bottleneck.

To address this challenge, our performance trace analysis uses a top-down approach where we start from the outermost parallelism level. As detailed in Section 5.2, our parallelism has an order of TP, CP, PP, and DP from inner to outer levels. We start the trace analysis from DP groups to identify the slowest one. Then we repeat this process for PP, CP, and TP groups to narrow down the range of the slow ranks step by step. Ultimately, after identifying the slow rank, we can check the detailed profiling trace of CPU, GPU compute, and GPU communication, to investigate the root cause whether it could be a software or hardware issue (e.g., a faulty GPU). We note that the method of identifying the bottleneck rank is similar to failure localization of distributed systems [39, 43] where the problematic host is not the same as the first one crashing and reporting errors. An automatic tool to analyze performance traces and identify the root cause of slowest rank will be helpful to performance debugging of Llama training systems.

Figure 8: Identify slow ranks in process groups.

6.2 Numerical issues in 4D Parallelism

Partitioning training data and model parameters in 4D parallelism inevitably changes numerical behaviors as floating-point additions are neither commutative nor associative. The use of low-precision data types such as BFloat16 (BF16) [13] further exacerbates the issue. Identifying and mitigating numerical gaps to ensure training stability is also an important design goal of the training system.

Differentiate numerical issues from implementation bugs: When developing 4D parallelism, it is important to distinguish whether the training loss behaviors are different as a result of numerical issues (e.g., different accumulation orders) or an implementation bug. The former can be mitigated by a higher precision accumulation order in certain parallelism implementations, whereas the latter needs further investigations to conclude the root cause. As parallelism splits computation into chunks and reduces partial results, it cannot achieve bit-wise matching results as the sequential version. To distinguish these two different reasons, we adopt an approach to split sequential version into the same accumulation order as the parallel one, and check whether results are bit-wise exactly matching. For example, we maintain a 2D parallelism (DP and TP) design with micro-batching to emulate the accumulation order of PP micro-batching. This reference baseline is helpful to confirm whether numerical gaps are from PP implementation bugs or accumulation order differences.

Accumulating gradients in FP32: With the aforementioned debugging process to identify numerical issues, we use FP32 accumulation for gradients and optimizer states to bridge numerical gaps although the rest parts for the model computation and communication are in BF16 formats. Specifically, we adopt FP32 for DP group reduce-scatter of gradients, and FP32 for accumulating the gradients of micro-batches in PP backwards. This accumulation precision aligns with hardware units where GEMM kernels accumulate partial results in FP32 for two BF16 input matrices. For backward computation, the accumulation happens along the batch size dimension where DP splits it into mini-batches and reduce-scatters

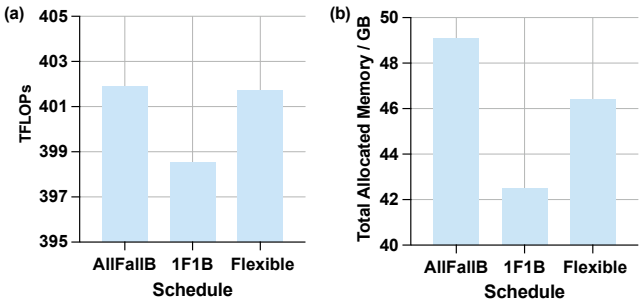


Figure 9: (a) Training TFLOPs for all forward all backward, 1F1B, and flexible PP. (b) Max allocated memory usage.

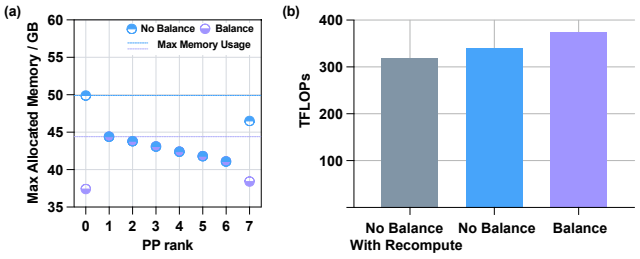


Figure 10: (a) Max allocated memory usage across PP ranks with and without workload balance. (b) Training throughput with balanced PP.

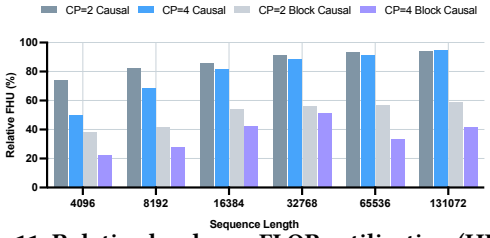


Figure 11: Relative hardware FLOPs utilization (HFU) over attention on single GPU.

gradients, and PP splits mini-batches further into micro-batches and accumulates gradients during PP backwards. In multimodal training, we further cast the image tokens that are sharded by all the cross-attention layers to FP32 so that the gradients are reduced across all the cross-attention in the backward in FP32 precision.

6.3 Memory Optimizations

When running 4D parallelism in large-scale systems, we find that 4D parallelism itself brings unique opportunities to improve memory efficiency other than splitting model parameters and input training data. For example, PP stage only needs forward output tensor metadata (i.e. tensor shape) to kick-off the backward pass but conventional PyTorch autograd engine is conservative in releasing memory with reference counting. To optimize memory usage in Llama 3 training system, we first profile the memory cost by the memory snapshot tool [1] to get a detailed memory allocation trace. Then we either develop a customized autograd operator to save tensor checkpoints during forward, or utilize existing autograd engine but release underlying tensor data by resizing the tensor storage manually. We note that these optimizations are helpful in our parallelism configurations to eliminate activation recomputation and avoid increasing PP or TP to fit the training into the memory thus helping with the training efficiency.

7 Evaluation

In this section, we evaluate PP and CP individually and also assess the end-to-end performance of 3D and 4D parallelism.

7.1 Pipeline Parallelism

We run small-scale experiments to demonstrate our optimizations. All experiments are conducted using a scaled-down Llama 3 405B model with the same model dimensions but fewer layers, using a sequence length of 8192.

7.1.1 Training throughput and memory comparison between all-forward-all-backward, 1F1B, and flexible PP. To compare between different schedules, we use a shrunk model with 26 layers and $pp = 4$, $bs = 12$. We reduce two layers from 28 to 26 for more balanced computation, as described in Section 3. All-forward-all-backward will run 12 micro-batches at once; 1F1B will run pp micro-batches in one round and 3 rounds per PP virtual stage in total; flexible PP will run 6 micro-batches in one round and 2 rounds in total. We present memory usage and TFLOPs for different schedules in Figure **??**. 1F1B has the lowest memory usage since it prioritizes backward pass, but achieves the lowest TFLOPs due to exposed P2Ps. All forward all backward has the highest TFLOPs with more micro-batches to hide exposed P2Ps, but results in the highest memory usage. Flexible PP is able to balance between memory usage and training throughput.

7.1.2 Balanced and non-balanced pipeline parallelism. In Llama 3 training, we use a vocabulary size of 128K. This results in a large embedding module on the first PP rank and a large output module on the last PP rank, leading to both computation and memory imbalance across PP ranks. By removing one layer from the first and last PP stages, memory is more balanced across PP ranks and training throughput is improved due to balanced computation. As is shown in Figure 10, the max allocated memory usage is reduced by 5GB and TFLOPs is improved by 6.5%. With this optimization, we are able to completely turn off activation recomputation [5] in Llama 3 training. With the shrunk model, balanced PP provides a 17.5% improvement on TFLOPs by avoiding recomputation.

7.2 Context Parallelism

As CP only introduces extra inter-GPU communication in the attention layers, we evaluate the efficiency of the attention layers to demonstrate the effectiveness and scalability of our CP solution.

We first study the efficiency and scalability of our CP solution by comparing with state-of-the-art attention kernels on GPU. In particular, we use flash-attention V2 [7] as our single GPU baseline, measure the hardware FLOPs utilization (HFU) [6], and normalize HFU of CP attention over flash-attention on a single GPU. As CP introduces communication between GPUs, we expect relative HFU to be smaller than 100%, and a higher relative HFU means a better

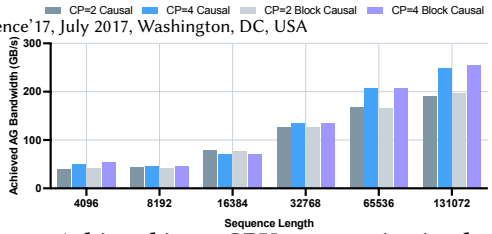


Figure 12: Achieved inter-GPU communication bandwidth of context parallelism all-gather.

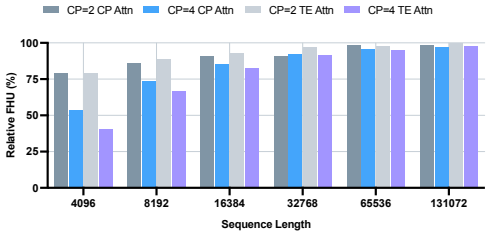


Figure 13: Relative hardware FLOPs utilization (HFU) comparison between context parallel attention (CP Attn) and TransformerEngine [27] attention (TE Attn).

efficiency of CP attention. We conduct experiments on H100 with HBM2e [26] to understand the scalability of CP attention in a lower memory bandwidth setup, and we expect a better scalability of CP attention in HBM3 as attention kernels are generally compute-bound while extra element-wise and communication overheads are usually memory or network bandwidth bound. We evaluate both $cp = 2$ and $cp = 4$ for different sequence lengths with full causal masks and block causal masks (i.e., document mask) where an average document length is 1K. We can observe from Figure 11 that (1) relative HFU of longer sequence length is higher (up to 95% relative HFUs for 128K sequence length), and (2) CP attention for block causal mask has a lower relative HFU. The observation (1) is consistent with our time complexity analysis in Section 4 that the elapsed time of CP communication scales linearly while computation scales quadratically with respect to sequence length. To further understand observation (2), we measure the achieved network bandwidth of CP all-gather communication, and detail results in Figure 12. In Figure 12, the achieved bandwidth of all-gather between causal and block causal masks are on par with each other. This indicates that a lower relative HFU of block causal mask comes from workload imbalance where our static sharding of tokens across CP does not align with document mask boundaries as shown in Figure 7.

In addition to the scalability study of CP attention, we also conduct experiments comparing our CP solution with Transformer Engine [27] (TE) in a branch before kicking off Llama 3 pre-training. TE attention adopts a computation-communication overlapped method similar to RingAttention [21]. In particular, it splits along the sequence length dimension into $2 \times cp$ chunks, assigns chunks to CP ranks to balance computation, iterates through chunks to compute partial attention results overlapped with P2P communication between adjacent ranks, and finally merges partial attention output results. In our forked branch of TE before kicking off Llama 3 training, it does not support variable sequence lengths, thus we only

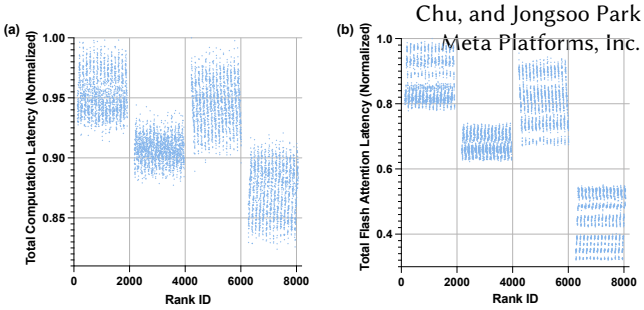


Figure 14: (a) The distribution of total computation time across all GPUs. (b) The distribution of the total time of attention kernels across all GPUs.

evaluate the full causal mask to compare with our CP attention solution. We conduct experiments on production hardware for Llama 3 training (i.e., H100 with HBM3) and Figure 13 shows relative HFU results. We observe from Figure 13 that TE has a slightly higher relative HFU in $cp = 2$, but both our CP and TE attention achieves relative HFU over 95% when sequence length is over 64K. We also observed that our CP attention consistently achieves a better relative HFU than TE attention for $cp = 4$, especially on 4K and 8K sequence lengths where we observe up to 13.53% better relative HFU. Different from our CP attention, the ring-style attention in TE has $O(cp)$ number of computation kernels where each computation kernel works on a chunk of $seq/(2 \times cp)$ tokens to compute partial results. When cp is large and sequence length is small, this ring-style attention in TE suffers from both fragmented compute kernels with lower compute efficiency, and the compute overheads of merging attention partial results.

7.3 End-to-end Performance

Llama 3 405B is trained on up to 16K H100 GPUs, each running at 700W TDP with 80GB HBM3, using Meta's Grand Teton AI server platform [22]. With the optimizations discussed in this paper, we achieve 400 TFLOPs/GPU for 8K sequence length, and 380 TFLOPs/GPU for 131K sequence length. We further detail the end-to-end performance and take-aways we observe for PP and CP as below.

7.3.1 Text model with 3D parallelism. We overlap FSDP's all-gather and reduce-scatter with other computation and communications, with only the first all-gather and last reduce-scatter exposed. P2P communications are exposed in warm-up forward and cool-down backward phases. Except for the shorter first and last model chunk due to having fewer transformer layers, forward and backward passes on regular model chunks are balanced across micro-batches and across PP stages, achieving 5% bubble ratio when per data parallel group $bs = 2 \times pp$ and 12% bubble ratio when $bs = pp$.

7.3.2 Long context text model with 4D parallelism. When training long context text model with 128K sequence length, we turn on $cp = 16$ so that each GPU rank still gets 8K sequence length as the base model with 3D parallelism. We conducted an extensive profiling of long context jobs with 8K GPUs, and found that the exposed latency of CP communication (all-gather in forward and reduce-scatter in backward) accounts for 7.64% of total elapsed time. However, an in-depth analysis reveals that, 65.75% of CP

exposed latency results from waiting for the slowest rank in the CP group to join the collective. The root cause comes from workload imbalances across all GPUs due to the document mask used in Llama 3 training [10], and this workload imbalance issue was more severe with longer sequence length and larger *cp*. Figure 14 (a) shows the total time of computation kernels across all GPUs, where the slowest rank spends 1.44× more time on computation than the fastest rank. A further analysis in Figure 14 (b) shows that this gap in total computation time fully comes from the difference in attention kernel time across GPUs, which means the imbalance due to the document mask contributes to the computation imbalance we have observed, thus making a large partition of CP exposed communication latency. Note that all parallel algorithms on CP to overlap CP communication and attention computation need to wait for the slowest CP rank to complete, leaving an upper-bound of 2.62% end-to-end performance improvement compared with our all-gather based CP solution.

8 Recommendations for Future Hardware

Hardware (HW) optimizations targeted for large-scale LLM training can be different from the ones for general AI workloads. Based on our experience with Llama training, we provide recommendations for future hardware optimized for LLM training.

8.1 Node level recommendations

For LLMs that require high dimensions of parallelism, it is essential to achieve high throughput from combination of accelerators and hosts.

Optimize compute efficiency for a wide range of shapes: It is not sufficient for a HW accelerator to provide high compute throughput for only very large shapes. Parallelisms will reduce the dimension of GEMMs, for example, PP reduces batch size and CP shards on sequence length. This can lead to lower arithmetic intensity operations, so we need to make sure that sufficient memory bandwidth is provided relative to compute throughput.

Higher HBM capacity can improve performance: Higher HBM capacity can increase the feasible hyper-parameter space for multi-dimensional parallelism, which can lead to higher overall performance. For example, sharding less in the tensor dimension leads to higher memory usage, but it also reduces TP communication overheads due to better amortization of communication relative to compute. Higher HBM capacity allows exploring all such options. However, the actual benefits depend on the model parameters, HW platform, and the cluster size.

Ensure sufficient CPU performance: The gen-over-gen performance improvement trend is significantly faster for accelerators than CPUs. Large-scale LLM training for future accelerators can become CPU-bound for multiple reasons. Scaling to large-clusters leads to smaller GEMM dimensions assigned to each accelerator, as stated above. Furthermore, model engineers can incorporate complex operations for exploration, which can lead to a sequence of lightweight kernels with relatively high host CPU overheads. These can lead to CPU times spent to prepare and launch the kernels becoming comparable to the accelerator runtimes unless addressed through HW and SW improvements.

Minimize performance variations and make DVFS deterministic: Parallelisms create synchronization among accelerators. If there are performance variations across different accelerators, the whole cluster performance will be determined by the slowest one. Furthermore, if different accelerators slow down at different times due to transient issues, the slow down will accumulate due to fine-grain synchronization across a large number of accelerators involved in the cross product of TP, CP, and PP domains. So, it is essential to make sure that dynamic voltage frequency scaling (DVFS) [34] policies are deterministic across accelerators to avoid transient slowdowns at different times.

8.2 Training cluster level recommendations

To further scale up training, we also need to consider how to connect nodes with efficient networks and make the best use of power for a data center.

Optimize network hierarchy: Scaling LLM training to 100K or more accelerators requires an efficient network with multiple levels of switches. Providing the same network bandwidth at each level of hierarchy will not be the most cost or power efficient design. Instead, a hierarchical network can be designed where the upper-level switches have less or oversubscribed bandwidth. It is essential to consider the requirements of different parallelism dimensions while determining these network parameters. We recommend co-designing the network parameters based on the anticipated workload requirements, taking into account model hyper-parameter as well as all dimensions of parallelism.

Ensure robust network performance: A slow down between two ranks (e.g., due to congestion or packet loss) can affect the whole cluster performance due to fine-grain parallelism across a large set of accelerators involved in the TP, CP, PP and DP communication. We need to make sure that the whole network operates at a consistent performance without transient slowdowns.

Prioritize power efficiency: It has been reported that future LLM training clusters are trending toward 100K GPUs or potentially more [32]. These large clusters are constrained by the total amount of power available in a data center region rather than the number of AI accelerators that can be procured. Therefore, an accelerator's effective performance per unit of power consumption (Perf/Watt) is as important as, or even more important than, its absolute performance.

9 Conclusion

This paper presents details of the training system for Llama 3 text and multimodal pre-training. We adopt 4D parallelism in this system for scaling out across up to 16K GPUs, and apply numerous optimizations to achieve high efficiency with the batch size constraint. We also target high flexibility when designing this system to support dynamic workloads for different training phases and heterogeneous model architectures. Moreover, we provide a methodology to debug performance and numerical issues at large scale. From our experience with Llama 3 training, we offer suggestions for future training node and cluster design. We hope that the details and insights shared in this paper could shed light on the directions of future model development and software-hardware co-design.

References

[1] Zachary DeVito Aaron Shi. 2023. Understanding GPU Memory 1: Visualizing All Allocations over Time. https://pytorch.org/blog/understanding-gpu-memory-1/

[2] Jean-Baptiste Alayrac, Jeff Donahue, Pauline Luc, Antoine Miech, Iain Barr, Yana Hasson, Karel Lenc, Arthur Mensch, Katherine Millican, Malcolm Reynolds, Roman Ring, Eliza Rutherford, Serkan Cabi, Tengda Han, Zhitao Gong, Sina Samangooei, Marianne Monteiro, Jacob L Menick, Sebastian Borgeaud, Andy Brock, Aida Nematzadeh, Sahand Sharifzadeh, Mikoł aj Bińkowski, Ricardo Barreira, Oriol Vinyals, Andrew Zisserman, and Karén Simonyan. 2022. Flamingo: a Visual Language Model for Few-Shot Learning. In Advances in Neural Information Processing Systems, S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh (Eds.), Vol. 35. Curran Associates, Inc., 23716–23736. https://proceedings.neurips.cc/paper_files/paper/2022/file/960a172bc7fbf0177ccccbb411a7d800-Paper-Conference.pdf

[3] Anthropic. 2024. Introducing the next generation of Claude. https://www.anthropic.com/news/claude-3-family

[4] Jinze Bai, Shuai Bai, Shusheng Yang, Shijie Wang, Sinan Tan, Peng Wang, Junyang Lin, Chang Zhou, and Jingren Zhou. 2023. Qwen-vl: A frontier large vision-language model with versatile abilities. arXiv preprint arXiv:2308.12966 (2023).

[5] Tianqi Chen, Bing Xu, Chiyuan Zhang, and Carlos Guestrin. 2016. Training deep nets with sublinear memory cost. arXiv preprint arXiv:1604.06174 (2016).

[6] Aakanksha Chowdhery, Sharan Narang, Jacob Devlin, Maarten Bosma, Gaurav Mishra, Adam Roberts, Paul Barham, Hyung Won Chung, Charles Sutton, Sebastian Gehrmann, Parker Schuh, Kensen Shi, Sashank Tsvyashchenko, Joshua Maynez, Abhishek Rao, Parker Barnes, Yi Tay, Noam Shazeer, Vinodkumar Prabhakaran, Emily Reif, Nan Du, Ben Hutchinson, Reiner Pope, James Bradbury, Jacob Austin, Michael Isard, Guy Gur-Ari, Pengcheng Yin, Toju Duke, Anselm Levskaya, Sanjay Ghemawat, Sunipa Dev, Henryk Michalewski, Xavier Garcia, Vedant Misra, Kevin Robinson, Liam Fedus, Denny Zhou, Daphne Ippolito, David Luan, Hyeontaek Lim, Barret Zoph, Alexander Spiridonov, Ryan Sepassi, David Dohan, Shivani Agrawal, Mark Omernick, Andrew M. Dai, Thanumalayan Sankaranarayana Pillai, Marie Pellat, Aitor Lewkowycz, Erica Moreira, Rewon Child, Oleksandr Polozov, Katherine Lee, Zongwei Zhou, Xuezhi Wang, Brennan Saeta, Mark Diaz, Orhan Firat, Michele Catasta, Jason Wei, Kathy Meier-Hellstern, Douglas Eck, Jeff Dean, Slav Petrov, and Noah Fiedel. 2024. PaLM: scaling language modeling with pathways. J. Mach. Learn. Res. 24, 1, Article 240 (March 2024), 113 pages.

[7] Tri Dao. 2024. FlashAttention-2: Faster Attention with Better Parallelism and Work Partitioning. In International Conference on Learning Representations (ICLR).

[8] Tri Dao, Daniel Y. Fu, Stefano Ermon, Atri Rudra, and Christopher Ré. 2022. FlashAttention: Fast and Memory-Efficient Exact Attention with IO-Awareness. In Advances in Neural Information Processing Systems (NeurIPS).

[9] DeepSeek-AI. 2024. DeepSeek-V3 Technical Report. arXiv preprint arXiv:2412.19437 (2024).

[10] Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, Amy Yang, Angela Fan, Anirudh Goyal, Anthony Hartshorn, Aobo Yang, Archi Mitra, Archie Sravankumar, Artem Korenev, Arthur Hinsvark, Arun Rao, Aston Zhang, Aurelien Rodriguez, Austen Gregerson, Ava Spataru, Baptiste Roziere, Bethany Biron, Binh Tang, Bobbie Chern, Charlotte Caucheteux, Chaya Nayak, Chloe Bi, Chris Marra, Chris McConnell, Christian Keller, Christophe Touret, Chunyang Wu, Corinne Wong, Cristian Canton Ferrer, Cyrus Nikolaidis, Damien Allonsius, Daniel Song, Danielle Pintz, Danny Livshits, Danny Wyatt, David Esiobu, Dhruv Choudhary, Dhruv Mahajan, Diego Garcia-Olano, Diego Perino, Dieuwke Hupkes, Egor Lakomkin, Ehab AlBadawy, Elina Lobanova, Emily Dinan, Eric Michael Smith, Filip Radenovic, Francisco Guzmán, Frank Zhang, Gabriel Synnaeve, Gabrielle Lee, Georgia Lewis Anderson, Govind Thattai, Graeme Nail, Gregoire Mialon, Guan Pang, Guillem Cucurell, Hailey Nguyen, Hannah Korevaar, Hu Xu, Hugo Touvron, Iliyan Zarov, Imanol Arrieta Ibarra, Isabel Kloumann, Ishan Misra, Ivan Evtimov, Jack Zhang, Jade Copet, Jaewon Lee, Jan Geffert, Jana Vranes, Jason Park, Jay Mahadeokar, Jeet Shah, Jelmer van der Linde, Jennifer Billock, Jenny Hong, Jenya Lee, Jeremy Fu, Jianfeng Chi, Jianyu Huang, Jiawen Liu, Jie Wang, Jiecao Yu, Joanna Bitton, Joe Spisak, Jongsoo Park, Joseph Rocca, Joshua Johnstun, Joshua Saxe, Junteng Jia, Kalyan Vasuden Alwala, Karthik Prasad, Kartikeya Upasani, Kate Plawiak, Ke Li, Kenneth Heafield, Kevin Stone, Khalid El-Arini, Krithika Iyer, Kshitiz Malik, Kuenley Chiu, Kunal Bhalla, Kushal Lakhotia, Lauren Rantala-Yeary, Laurens van der Maaten, Lawrence Chen, Liang Tan, Liz Jenkins, Louis Martin, Lovish Madaan, Lubo Malo, Lukas Blecher, Lukas Landzaat, Luke de Oliveira, Madeline Muzzi, Mahesh Pasupuleti, Mannat Singh, Manohar Paluri, Marcin Kardas, Maria Tsimpoukelli, Mathew Oldham, Mathieu Rita, Maya Pavlova, Melanie Kambadur, Mike Lewis, Min Si, Mitesh Kumar Singh, Mona Hassan, Naman Goyal, Narjes Torabi, Nikolay Bashlykov, Nikolay Bogoychev, Niladri Chatterji, Ning Zhang, Olivier Duchenne, Onur Çelebi, Patrick Alrassy, Pengchuan Zhang, Pengwei Li, Petar Vasic, Peter Weng, Prajjwal Bhargava, Pratik Dubal, Praveen Krishnan, Punit Singh Koura, Puxin Xu, Qing He, Qingxiao Dong, Ragavan Srinivasan, Raj Ganapathy, Ramon Calderer, Ricardo Silveira Cabral, Robert Stojnic, Roberta Raileanu, Rohan Maheswari, Rohit Girdhar, Rohit Patel, Romain Sauvestre, Ronnie Polidoro, Roshan Sumbaly, Ross Taylor, Ruan Silva, Rui Hou, Rui Wang, Saghar Hosseini, Sahana Chennabasappa, Sanjay Singh, Sean Bell, Seohyun Sonia Kim, Sergey Edunov, Shaoliang Nie, Sharan Narang, Sharath Raparthy, Sheng Shen, Shengye Wan, Shruti Bhosale, Shun Zhang, Simon Vandenhende, Soumya Batra, Spencer Whitman, Sten Sootla, Stephane Collot, Suchin Gururangan, Sydney Borodinsky, Tamar Herman, Tara Fowler, Tarek Sheasha, Thomas Georgiou, Thomas Scialom, Tobias Speckbacher, Todor Mihaylov, Tong Xiao, Ujjwal Karn, Vedanuj Goswami, Vibhor Gupta, Vignesh Ramanathan, Viktor Kerkez, Vincent Gonguet, Virginie Do, Vish Vogeti, Vítor Albiero, Vladan Petrovic, Weiwei Chu, Wenhan Xiong, Wenyin Fu, Whitney Meers, Xavier Martinet, Xiaodong Wang, Xiaofang Wang, Xiaoqing Ellen Tan, Xide Xia, Xinfeng Xie, Xuchao Jia, Xuewei Wang, Yaelle Goldschlag, Yashesh Gaur, Yasmine Babaei, Yi Wen, Yiwen Song, Yuchen Zhang, Yue Li, Yuning Mao, Zacharie Delpierre Coudert, Zheng Yan, Zhengxing Chen, Zoe Papakipos, Aaditya Singh, Aayushi Srivastava, Abha Jain, Adam Kelsey, Adam Shajnfeld, Adithya Gangidi, Adolfo Victoria, Ahuva Goldstand, Ajay Menon, Ajay Sharma, Alex Boesenberg, Alexei Baevski, Allie Feinstein, Amanda Kallet, Amit Sangani, Amos Teo, Anam Yunus, Andrei Lupu, Andres Alvarado, Andrew Caples, Andrew Gu, Andrew Ho, Andrew Poulton, Andrew Ryan, Ankit Ramchandani, Annie Dong, Annie Franco, Anuj Goyal, Aparajita Saraf, Arkabandhu Chowdhury, Ashley Gabriel, Ashwin Bharambe, Assaf Eisenman, Azadeh Yazdan, Beau James, Ben Maurer, Benjamin Leonhardi, Bernie Huang, Beth Loyd, Beto De Paola, Bhargavi Paranjape, Bing Liu, Bo Wu, Boyu Ni, Braden Hancock, Bram Wasti, Brandon Spence, Brani Stojkovic, Brian Gamido, Britt Montalvo, Carl Parker, Carly Burton, Catalina Mejia, Ce Liu, Changhan Wang, Changkyu Kim, Chao Zhou, Chester Hu, Ching-Hsiang Chu, Chris Cai, Chris Tindal, Christoph Feichtenhofer, Cynthia Gao, Damon Civin, Dana Beaty, Daniel Kreymer, Daniel Li, David Adkins, David Xu, Davide Testuggine, Delia David, Devi Parikh, Diana Liskovich, Didem Foss, Dingkang Wang, Duc Le, Dustin Holland, Edward Dowling, Eissa Jamil, Elaine Montgomery, Eleonora Presani, Emily Hahn, Emily Wood, Eric-Tuan Le, Erik Brinkman, Esteban Arcaute, Evan Dunbar, Evan Smothers, Fei Sun, Felix Kreuk, Feng Tian, Filippos Kokkinos, Firat Ozgenel, Francesco Caggioni, Frank Kanayet, Frank Seide, Gabriela Medina Florez, Gabriella Schwarz, Gada Badeer, Georgia Swee, Gil Halpern, Grant Herman, Grigory Sizov, Guangyi, Zhang, Guna Lakshminarayanan, Hakan Inan, Hamid Shojanazeri, Han Zou, Hannah Wang, Hanwen Zha, Haroun Habeeb, Harrison Rudolph, Helen Suk, Henry Aspegren, Hunter Goldman, Hongyuan Zhan, Ibrahim Damlaj, Igor Molybog, Igor Tufanov, Ilias Leontiadis, Irina-Elena Veliche, Itai Gat, Jake Weissman, James Geboski, James Kohli, Janice Lam, Japhet Asher, Jean-Baptiste Gaya, Jeff Marcus, Jeff Tang, Jennifer Chan, Jenny Zhen, Jeremy Reizenstein, Jeremy Teboul, Jessica Zhong, Jian Jin, Jingyi Yang, Joe Cummings, Jon Carvill, Jon Shepard, Jonathan McPhie, Jonathan Torres, Josh Ginsburg, Junjie Wang, Kai Wu, Kam Hou U, Karan Saxena, Kartikay Khandelwal, Katayoun Zand, Kathy Matosich, Kaushik Veeraraghavan, Kelly Michelena, Keqian Li, Kiran Jagadeesh, Kun Huang, Kunal Chawla, Kyle Huang, Lailin Chen, Lakshya Garg, Lavender A, Leandro Silva, Lee Bell, Lei Zhang, Liangpeng Guo, Licheng Yu, Liron Moshkovich, Luca Wehrstedt, Madian Khabsa, Manav Avalani, Manish Bhatt, Martynas Mankus, Matan Hasson, Matthew Lennie, Matthias Reso, Maxim Groshev, Maxim Naumov, Maya Lathi, Meghan Keneally, Miao Liu, Michael L. Seltzer, Michal Valko, Michelle Restrepo, Mihir Patel, Mik Vyatskov, Mikayel Samvelyan, Mike Clark, Mike Macey, Mike Wang, Miquel Jubert Hermoso, Mo Metanat, Mohammad Rastegari, Munish Bansal, Nandhini Santhanam, Natascha Parks, Natasha White, Navyata Bawa, Nayan Singhal, Nick Egebo, Nicolas Usunier, Nikhil Mehta, Nikolay Pavlovich Laptev, Ning Dong, Norman Cheng, Oleg Chernoguz, Olivia Hart, Omkar Salpekar, Ozlem Kalinli, Parkin Kent, Parth Parekh, Paul Saab, Pavan Balaji, Pedro Rittner, Philip Bontrager, Pierre Roux, Piotr Dollar, Polina Zvyagina, Prashant Ratanchandani, Pritish Yuvraj, Qian Liang, Rachad Alao, Rachel Rodriguez, Rafi Ayub, Raghotham Murthy, Raghu Nayani, Rahul Mitra, Rangaprabhu Parthasarathy, Raymond Li, Rebekkah Hogan, Robin Battey, Rocky Wang, Russ Howes, Ruty Rinott, Sachin Mehta, Sachin Siby, Sai Jayesh Bondu, Samyak Datta, Sara Chugh, Sara Hunt, Sargun Dhillon, Sasha Sidorov, Satadru Pan, Saurabh Mahajan, Saurabh Verma, Seiji Yamamoto, Sharadh Ramaswamy, Shaun Lindsay, Shaun Lindsay, Sheng Feng, Shenghao Lin, Shengxin Cindy Zha, Shishir Patil, Shiva Shankar, Shuqiang Zhang, Shuqiang Zhang, Sinong Wang, Sneha Agarwal, Soji Sajuyigbe, Soumith Chintala, Stephanie Max, Stephen Chen, Steve Kehoe, Steve Satterfield, Sudarshan Govindaprasad, Sumit Gupta, Summer Deng, Sungmin Cho, Sunny Virk, Suraj Subramanian, Sy Choudhury, Sydney Goldman, Tal Remez, Tamar Glaser, Tamara Best, Thilo Koehler, Thomas Robinson, Tianhe Li, Tianjun Zhang, Tim Matthews, Timothy Chou, Tzook Shaked, Varun Vontimitta, Victoria Ajayi, Victoria Montanez, Vijai Mohan, Vinay Satish Kumar, Vishal Mangla, Vlad Ionescu, Vlad Poenaru, Vlad Tiberiu Mihailescu, Vladimir Ivanov, Wei Li, Wenchen Wang, Wenwen Jiang, Wes Bouaziz, Will Constable, Xiaocheng Tang, Xiaojian Wu, Xiaolan Wang, Xilun Wu, Xinbo Gao, Yaniv Kleinman, Yanjun Chen, Ye Hu, Ye Jia, Ye Qi, Yenda Li, Yilin Zhang, Ying Zhang, Yossi Adi, Youngjin Nam, Yu, Wang, Yu Zhao, Yuchen Hao, Yundi Qian, Yunlu Li, Yuzi He, Zach Rait, Zachary DeVito, Zef Rosnbrick, Zhaoduo Wen, Zhenyu Yang, Zhiwei Zhao, and Zhiyu Ma. 2024. The Llama 3 Herd of Models. arXiv:2407.21783 [cs.AI] https://arxiv.org/abs/2407.21783

[11] 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. 2019. GPipe: Efficient Training of Giant Neural Networks using Pipeline Parallelism. arXiv:1811.06965 [cs.CV] https://arxiv.org/abs/1811.06965

[12] Ziheng Jiang, Haibin Lin, Yinmin Zhong, Qi Huang, Yangrui Chen, Zhi Zhang, Yanghua Peng, Xiang Li, Cong Xie, Shibiao Nong, Yulu Jia, Sun He, Hongmin Chen, Zhihao Bai, Qi Hou, Shipeng Yan, Ding Zhou, Yiyao Sheng, Zhuo Jiang, Haohan Xu, Haoran Wei, Zhang Zhang, Pengfei Nie, Leqi Zou, Sida Zhao, Liang Xiang, Zherui Liu, Zhe Li, Xiaoying Jia, Jianxi Ye, Xin Jin, and Xin Liu. 2025. MegaScale: Scaling Large Language Model Training to More Than 10,000 GPUs. In Proceedings of the 21st USENIX Symposium on Networked Systems Design and Implementation (Santa Clara, CA, USA) (NSDI'24). USENIX Association, USA, Article 41, 16 pages.

[13] Dhiraj Kalamkar, Dheevatsa Mudigere, Naveen Mellempudi, Dipankar Das, Kunal Banerjee, Sasikanth Avancha, Dharma Teja Vooturi, Nataraj Jammalamadaka, Jianyu Huang, Hector Yuen, Jiyan Yang, Jongsoo Park, Alexander Heinecke, Evangelos Georganas, Sudarshan Srinivasan, Abhisek Kundu, Misha Smelyanskiy, Bharat Kaul, and Pradeep Dubey. 2019. A Study of BFLOAT16 for Deep Learning Training. (2019). arXiv:1905.12322 [cs.LG] https://arxiv.org/abs/1905.12322

[14] Vijay Korthikanti, Jared Casper, Sangkug Lym, Lawrence McAfee, Michael Andersch, Mohammad Shoeybi, and Bryan Catanzaro. 2022. Reducing Activation Recomputation in Large Transformer Models. arXiv:2205.05198 [cs.LG] https://arxiv.org/abs/2205.05198

[15] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. 2012. ImageNet Classification with Deep Convolutional Neural Networks. In Advances in Neural Information Processing Systems, F. Pereira, C.J. Burges, L. Bottou, and K.Q. Weinberger (Eds.), Vol. 25. Curran Associates, Inc. https://proceedings.neurips.cc/paper_files/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf

[16] Joel Lamy-Poirier. 2023. Breadth-First Pipeline Parallelism. arXiv:2211.05953 [cs.DC] https://arxiv.org/abs/2211.05953

[17] 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. (2020). arXiv:2006.15704 [cs.DC] https://arxiv.org/abs/2006.15704

[18] Yujia Li, David Choi, Junyoung Chung, Nate Kushman, Julian Schrittwieser, Rémi Leblond, Tom Eccles, James Keeling, Felix Gimeno, Agustin Dal Lago, Thomas Hubert, Peter Choy, Cyprien de Masson d'Autume, Igor Babuschkin, Xinyun Chen, Po-Sen Huang, Johannes Welbl, Sven Gowal, Alexey Cherepanov, James Molloy, Daniel J. Mankowitz, Esme Sutherland Robson, Pushmeet Kohli, Nando de Freitas, Koray Kavukcuoglu, and Oriol Vinyals. 2022. Competition-level code generation with AlphaCode. Science 378, 6624 (2022), 1092–1097. https://doi.org/10.1126/science.abq1158 arXiv:https://www.science.org/doi/pdf/10.1126/science.abq1158

[19] Haotian Liu, Chunyuan Li, Qingyang Wu, and Yong Jae Lee. 2024. Visual instruction tuning. Advances in neural information processing systems 36 (2024).

[20] Hao Liu, Matei Zaharia, and Pieter Abbeel. 2023. Ring Attention with Blockwise Transformers for Near-Infinite Context. arXiv:2310.01889 [cs.CL] https://arxiv.org/abs/2310.01889

[21] Hao Liu, Matei Zaharia, and Pieter Abbeel. 2023. Ring Attention with Blockwise Transformers for Near-Infinite Context. arXiv:2310.01889 [cs.CL] https://arxiv.org/abs/2310.01889

[22] Meta. 2022. Meta open compute project, grand teton ai platform. https://engineering.fb.com/2022/10/18/open-source/ocp-summit-2022-grand-teton

[23] Meta. 2024. Introducing Llama 3.1: Our most capable models to date. https://ai.meta.com/blog/meta-llama-3-1/

[24] Meta. 2024. Introducing Meta Llama 3: The most capable openly available LLM to date. https://ai.meta.com/blog/meta-llama-3/

[25] Deepak Narayanan, Mohammad Shoeybi, Jared Casper, Patrick LeGresley, Mostofa Patwary, Vijay Anand 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 International Conference for High Performance Computing, Networking, Storage and Analysis, SC 2021, St. Louis, Missouri, USA, November 14-19, 2021. ACM. https://arxiv.org/abs/2104.04473

[26] NVIDIA. 2022. NVIDIA Hopper Architecture In-Depth. https://developer.nvidia.com/blog/nvidia-hopper-architecture-in-depth/

[27] NVIDIA. 2024. NVIDIA TransformerEngine. https://github.com/NVIDIA/TransformerEngine

[28] OpenAI. 2022. Introducing ChatGPT. https://openai.com/index/chatgpt/

[29] Qwen Team. 2024. Qwen2.5 Technical Report. arXiv preprint arXiv:2412.15115 (2024).

[30] Samyam Rajbhandari, Jeff Rasley, Olatunji Ruwase, and Yuxiong He. 2020. ZeRO: Memory Optimizations Toward Training Trillion Parameter Models. arXiv:1910.02054 [cs.LG] https://arxiv.org/abs/1910.02054

[31] 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 [cs.DC] https://arxiv.org/abs/2101.06840

[32] SemiAnalysis. 2025. 100,000 H100 Clusters: Power, Network Topology, Ethernet vs InfiniBand, Reliability, Failures, Checkpointing. https://semianalysis.com/2024/06/17/100000-h100-clusters-power-network/

[33] Mohammad Shoeybi, Mostofa Patwary, Raul Puri, Patrick LeGresley, Jared Casper, and Bryan Catanzaro. 2020. Megatron-LM: Training Multi-Billion Parameter Language Models Using Model Parallelism. arXiv:1909.08053 [cs.CL] https://arxiv.org/abs/1909.08053

[34] Zhenheng Tang, Yuxin Wang, Qiang Wang, and Xiaowen Chu. 2019. The Impact of GPU DVFS on the Energy and Performance of Deep Learning: an Empirical Study. In Proceedings of the Tenth ACM International Conference on Future Energy Systems. 315–325.

[35] Jakub Tarnawski, Deepak Narayanan, and Amar Phanishayee. 2021. Piper: Multidimensional Planner for DNN Parallelization. In Advances in Neural Information Processing Systems 34: Annual Conference on Neural Information Processing Systems 2021, NeurIPS 2021, December 6-14, 2021, virtual. 24829–24840.

[36] Team Gemini. 2023. Gemini: A Family of Highly Capable Multimodal Models. arXiv preprint arXiv:2312.11805 (2023).

[37] Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, Aurelien Rodriguez, Armand Joulin, Edouard Grave, and Guillaume Lample. 2023. LLaMA: Open and Efficient Foundation Language Models. (2023). arXiv:2302.13971 [cs.CL] https://arxiv.org/abs/2302.13971

[38] Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, Dan Bikel, Lukas Blecher, Cristian Canton Ferrer, Moya Chen, Guillem Cucurull, David Esiobu, Jude Fernandes, Jeremy Fu, Wenyin Fu, Brian Fuller, Cynthia Gao, Vedanuj Goswami, Naman Goyal, Anthony Hartshorn, Saghar Hosseini, Rui Hou, Hakan Inan, Marcin Kardas, Viktor Kerkez, Madian Khabsa, Isabel Kloumann, Artem Korenev, Punit Singh Koura, Marie-Anne Lachaux, Thibaut Lavril, Jenya Lee, Diana Liskovich, Yinghai Lu, Yuning Mao, Xavier Martinet, Todor Mihaylov, Pushkar Mishra, Igor Molybog, Yixin Nie, Andrew Poulton, Jeremy Reizenstein, Rashi Rungta, Kalyan Saladi, Alan Schelten, Ruan Silva, Eric Michael Smith, Ranjan Subramanian, Xiaoqing Ellen Tan, Binh Tang, Ross Taylor, Adina Williams, Jian Xiang Kuan, Puxin Xu, Zheng Yan, Iliyan Zarov, Yuchen Zhang, Angela Fan, Melanie Kambadur, Sharan Narang, Aurelien Rodriguez, Robert Stojnic, Sergey Edunov, and Thomas Scialom. 2023. Llama 2: Open Foundation and Fine-Tuned Chat Models. arXiv:2307.09288 [cs.CL] https://arxiv.org/abs/2307.09288

[39] W Eric Wong, Ruizhi Gao, Yihao Li, Rui Abreu, and Franz Wotawa. 2016. A survey on software fault localization. IEEE Transactions on Software Engineering 42, 8 (2016), 707–740.

[40] Zhiyu Wu, Xiaokang Chen, Zizheng Pan, Xingchao Liu, Wen Liu, Damai Dai, Huazuo Gao, Yiyang Ma, Chengyue Wu, Bingxuan Wang, Zhenda Xie, Yu Wu, Kai Hu, Jiawei Wang, Yaofeng Sun, Yukun Li, Yishi Piao, Kang Guan, Aixin Liu, Xin Xie, Yuxiang You, Kai Dong, Xingkai Yu, Haowei Zhang, Liang Zhao, Yisong Wang, and Chong Ruan. 2024. DeepSeek-VL2: Mixture-of-Experts Vision-Language Models for Advanced Multimodal Understanding. (2024). arXiv:2412.10302 [cs.CV] https://arxiv.org/abs/2412.10302

[41] Wenhan Xiong, Jingyu Liu, Igor Molybog, Hejia Zhang, Prajjwal Bhargava, Rui Hou, Louis Martin, Rashi Rungta, Karthik Abinav Sankararaman, Barlas Oguz, Madian Khabsa, Han Fang, Yashar Mehdad, Sharan Narang, Kshitiz Malik, Angela Fan, Shruti Bhosale, Sergey Edunov, Mike Lewis, Sinong Wang, and Hao Ma. 2023. Effective Long-Context Scaling of Foundation Models. arXiv:2309.16039 [cs.CL] https://arxiv.org/abs/2309.16039

[42] Hu Xu, Saining Xie, Xiaoqing Ellen Tan, Po-Yao Huang, Russell Howes, Vasu Sharma, Shang-Wen Li, Gargi Ghosh, Luke Zettlemoyer, and Christoph Feichtenhofer. 2023. Demystifying clip data. arXiv preprint arXiv:2309.16671 (2023).

[43] Yongle Zhang, Kirk Rodrigues, Yu Luo, Michael Stumm, and Ding Yuan. 2019. The inflection point hypothesis: a principled debugging approach for locating the root cause of a failure. In Proceedings of the 27th ACM Symposium on Operating Systems Principles. 131–146.

[44] 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. 2023. PyTorch FSDP: Experiences on Scaling Fully Sharded Data Parallel. arXiv:2304.11277 [cs.DC] https://arxiv.org/abs/2304.11277

[45] 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 2022, Carlsbad, CA, USA, July 11-13, 2022. USENIX Association, 559–578.