Tale of Two Cs: Computation vs. Communication Scaling for Future Transformers on Future Hardware

Suchita Pati[*†] Shaizeen Aga[†] Mahzabeen Islam[†] Nuwan Jayasena[†] Matthew D. Sinclair[*†]

[†]*Advanced Micro Devices, Inc.*
[*]*University of Wisconsin-Madison*

Abstract

Scaling neural network models has delivered dramatic quality gains across ML problems. However, this scaling also increased the reliance on efficient distributed training techniques. Accordingly, like other distributed computing scenarios, it is important to understand how compute and communication will scale relative to one another as models scale and hardware evolves? A careful study which answers this question can better guide the design of future systems which can efficiently train future large models.

*Accordingly, we comprehensively analyze compute vs. communication (**Comp-vs.-Comm**) scaling for future Transformer models on future hardware, across multiple axes (algorithmic, empirical, hardware evolution). First, our algorithmic analysis shows that compute generally enjoys an edge over communication as models scale. However, these trends are being stressed since device memory capacity scales much slower than model size. We quantify this edge by empirically studying how Comp-vs.-Comm scales for future models on future hardware. To avoid profiling numerous Transformer models across many setups, we extract execution regions and project costs using operator models. This allows a spectrum (hundreds) of future model/hardware scenarios to be accurately studied ($<$ 15% error) and reduces profiling costs by 2100×. Our experiments show that communication will be a significant portion (40-75%) of runtime as models and hardware evolve. Moreover, communication that is often hidden by overlapped computation in today's models cannot be hidden in future, larger models. Overall, this work highlights communication's increasingly large role as models scale, discusses promising techniques to potentially tackle communication, and discusses how our analysis influences their potential improvements.*

1. Introduction

In recent years, machine learning (ML) and deep neural networks (DNNs) have transformed society, including significant improvements in accuracy on tasks including speech recognition [77], image classification [28, 35, 38, 66, 71, 72], machine translation [27], autonomous agents [39], and language processing [12, 18, 53]. This transformative effect has been enabled by a virtuous synergy of (1) more efficient hardware, (2) larger datasets, and (3) algorithmic advances (including exponential model size growth) that further benefit from more efficient hardware and larger datasets. However, models are scaling much more rapidly (1000×) than per-GPU memory scaling (5×) [56]. This, along with the increasing computational demands as models scale, has increased the reliance on distributed training: multiple accelerators (e.g., GPUs) harness their collective memory capacities and compute capabilities to collaboratively train a DNN. Consequently, it is important to understand *how compute and communication in distributed training will scale relative to one another as DNNs scale and hardware evolves.*

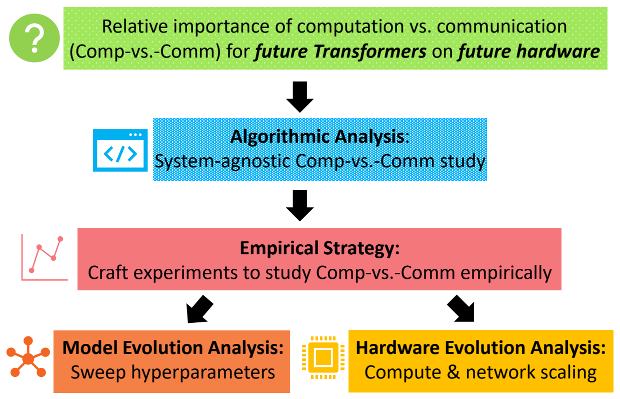

Relative importance of computation vs. communication (Comp-vs.-Comm) for future Transformers on future hardware
Algorithmic Analysis: System-agnostic Comp-vs.-Comm study
Empirical Strategy: Craft experiments to study Comp-vs.-Comm empirically
Model Evolution Analysis: Sweep hyperparameters
Hardware Evolution Analysis: Compute & network scaling

Figure 1: Overview of Comp-vs.-Comm analysis.

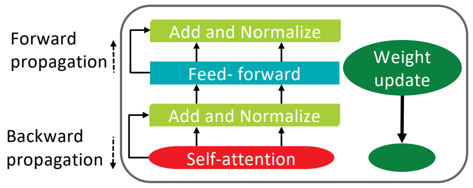
This work performs this analysis, which we term **Comp-vs.-Comm**, to better guide the design of future systems to more efficiently train future large models. We perform a comprehensive multi-axial (algorithmic, empirical, hardware evolution) Comp-vs.-Comm analysis for future Transformer models on future hardware. Although many different DNN models are currently used, Transformers have arisen to be a candidate general purpose architecture [11] capable of tackling multiple tasks across different modalities (e.g., text, vision, audio). Furthermore, Transformers have shown considerable strides in artificial general intelligence. For example, recently researchers trained a single 1.2B Transformer to perform hundreds of robotics, simulated environments, and vision and language tasks [59]. Thus, given their ubiquity, growing importance, and capabilities, we focus on Transformers.

Figure 1 summarizes our approach. We first perform an algorithmic analysis of compute and communication operations in Transformer models. This analysis provides a system-agnostic view of Comp-vs.-Comm scaling, which is important given the wide variety of system/infrastructure capabilities, ranging from standalone accelerators to accelerator clusters with state-of-art interconnects. Moreover, this analysis provides a strong foundation to empirically study Comp-vs.-Comm without significant overhead. Our algorithmic analysis shows that the complexity of compute operations is often higher than communication volume (data size). We call this ***compute's edge*** over communication. A compute-dominated execution profile is often a positive edge because (a) traditionally, and especially for accelerators, compute (FLOPS) scaling has received considerably more attention than communication (bandwidth) scaling, and (b) often algorithmic/system optimizations are employed to overlap communication with useful compute. Thus, compute's edge also helps hide communication costs. Compute enjoys this edge for both serialized and overlapped compute and communication scenarios – both of which occur in distributed training. However, model scaling and memory capacity trends are stressing this edge.

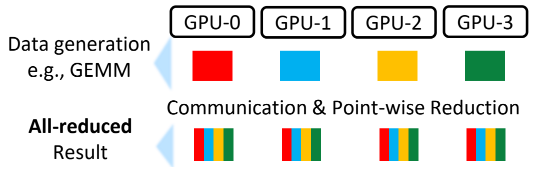
To understand how compute's edge may be impacted by future models and future hardware, we empirically study Comp-vs.-Comm scaling. This approach has several challenges, including requiring studying many model/hardware evolution scenarios, each of which incurs significant profiling costs. Our empirical strategy addresses these challenges by (a) designing controlled experiments (guided by our algorithmic analysis), (b) executing only certain regions-of-interest (ROIs) to reduce profiling costs, and (c) using operator-level models for training operations, which we show accurately capture runtime trends for varying hyperparameters. These enable us to study hundreds of future models/hardware scenarios at 2100× lower profiling costs. Further, given the generality of our empirical strategy we discuss how to apply Comp-vs.-Comm scaling analysis to future Transformer models.

Our empirical strategy-driven experiments back-up the conclusions from our algorithmic analysis. Specifically, we find that the compute's edge over communication is stressed: up to 50% of a future Transformer's training time will be spent communicating data. Furthermore, communication that is overlapped or hidden today can exceed the compute time in future models, further increasing communication's proportion. Moreover, if past hardware evolution trends on scaling of compute capability vis-a-vis communication bandwidth continue, communication will become an even bigger bottleneck ($>75\%$ of training execution) on future systems. We make the following contributions:

- As models scale and training becomes increasingly distributed, a careful understanding of compute vs. communication scaling is paramount to effectively designing future systems. Accordingly, we provide a comprehensive multi-axial analysis (algorithmic, empirical, hardware evolution) of Comp-vs.-Comm scaling for future Transformers on future hardware.
- Using algorithmic/system-agnostic analysis we show that while compute has had an edge over communication (both in serialized and overlapped scenarios), models are scaling faster than memory capacity – stressing this edge.



(a) Compute: Transformer block.


GPU-0
GPU-1
GPU-2
GPU-3
Data generation e.g., GEMM
Communication & Point-wise Reduction
All-reduced Result

(b) Communication: All-reduce.

Figure 2: Distributed Transformer Components.

- We devise a strategy for a practical empirical Comp-vs.-Comm study for future models. It extracts execution ROIs and accurately ($<15\%$ error) projects future models' operation times to reduce profiling costs: 2100× faster than executing the models.
- Our experiments demonstrate how communication will play an increasingly large role as models scale: up to 50% of training time will be spent on communication. Furthermore, if past compute vs. network scaling trends continue, communication can end up being the critical bottleneck in distributed training: up to 75% of training time will be spent on communication and some previously hidden communication costs will be exposed.
- We discuss techniques and technologies that tackle communication and how our analysis influences their improvements.

Overall, our work highlights communication's increasingly large role as Transformer models scale.

2. Background and Motivation

This section summarizes Transformers [73], their distributed training, and the need for a Comp-vs.-Comm scaling study.

2.1. Transformers: Building Blocks of Future Models

Transformers [73] have become a popular general-purpose architecture for a wide range of tasks/domains. Recent work has shown that many different modalities are using Transformers as their base model (e.g., 41% of text, 22% of image) [11]. The basic building block of these Transformers is an *encoder* or *decoder* which are repeated multiple times. As shown in Figure 2a, each block contains an attention layer and a fully connected (FC) layer, both of which are followed by a residual connection and layer normalization. These layers manifest as matrix multiplication operations (GEMMs) followed by a few element-wise operations, which are often fused [20, 23,

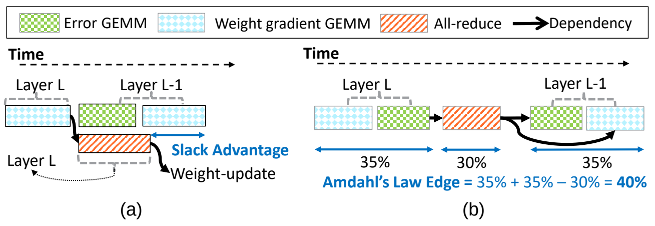


Figure 3: All-reduce in (a) data & (b) tensor parallelism.

69, 75] with the GEMMs. The encoder and decoder blocks are similar, except the decoder's attention GEMM input is masked, which causes different computational inference behavior but does not affect training (our focus).

The evolution of Transformer models has largely focused on changing Transformer block type (encoder vs. decoder), increasing the number of Transformer blocks, and/or increasing layer widths. This is true for all Transformer models; starting with BERT [18] (0.3 billion parameters), to its most recent successor, MT-NLG (540 billion parameters), and many others in between [12, 17, 36, 41, 53, 65, 70, 78]. Thus, while Transformer models have become larger with different hyperparameters, their fundamental computational components are largely the same. Therefore, we use BERT as our baseline and change its hyperparameters to study larger Transformer models. However, our methodology can be extended to other DNNs (see Section 6, *Other DNNs*).

2.2. Distributed Training for Large Scale Training

Most Transformer models employ distributed techniques and use multiple accelerators (e.g., GPUs) collaboratively to train. Distributing training is often necessary since the model's memory capacity requirements are too large for one device. The aggregate computational capacity of multiple devices also accelerates training by operating on large input datasets in parallel. Thus, since Transformers and their datasets (usually large corpuses of unlabeled text) have increased several orders of magnitude in size (Section 3.5), distributed techniques have not only become de facto but also require increasingly larger numbers of devices. This scale will only increase for future models (Section 4.3.2). However, our analysis is also applicable to fine-tuning and inference (see Section 6, *Fine-tuning & Inference*).

2.3. Distributed Training Techniques and Associated Communication

All distributed training techniques have associated *communication* between devices to transfer (e.g., in pipeline parallelism [31]), reduce (e.g., in tensor [65] and data parallelism), aggregate (e.g., in ZeRO-based optimizations [56]), or exchange (e.g., in expert parallelism [33]) information. These communication patterns are handled by *collectives* such as *all-reduce*, *all-gather*, *all-to-all*. We focus on all-reduce, the collective used in two of the most effective and widely adopted distributed techniques (Section 3.1): data and tensor parallelism. We discuss the communication impact of other techniques in Section 6, *Beyond DP & TP*.

2.3.1. All-reduce Communication Flavors. Figure 2b shows how the all-reduce (AR) collective reduces data generated by all participating devices and broadcasts the reduced data to them. AR has different implementations optimized for different system topologies. While the AR collective remains the same, involving both communication and compute (e.g., element-wise summation), in both data parallelism (DP) and tensor parallelism (TP) setups its usage and thus its impact on the overall training behavior differs.

2.3.2. Data Parallelism (DP) & Slack Advantage. DP effectively increases the training input batch size by duplicating the model on all devices, with each operating on a disjoint set of inputs. To keep the model copies in sync, the devices all-reduce their weight gradients during the backward training pass. This all-reduce of gradients from one layer can happen asynchronously with the gradient calculation of another layer (Figure 3(a)). Thus, the associated communication can potentially be overlapped and hidden by computations. However, complete overlap is only possible if the execution of computations equals or exceeds that of communication. We call this difference in overlapped compute and communication executions of each layer compute's *slack advantage* (Figure 3(a)).

2.3.3. Tensor/Horizontal Parallel (TP) & Amdahl's Law Edge. TP effectively increases the memory capacity available to a model by splitting it across devices (illustrated in Figure 4). It splits a model layer across devices such that each device holds and thus operates on a subset of layer parameters. This slicing causes each device to generate only a partial layer activation (and error) during training's forward (and backward) passes (light blue matrices in Figure 4(b)), which require an all-reduce to generate the final layer output (deep blue matrices in Figure 4(b)). Furthermore, a layer's forward and backward executions are dependent on another layer's all-reduce of activations and errors. Thus, unlike DP, in TP compute and communication are not asynchronous and communication is on the critical path of model execution (Figure 3(b)). We call the difference in compute and serialized communication executions compute's *Amdahl's Law edge* (Figure 3(b)).

2.4. Why Study Evolution of Compute vs. Communication Scaling

Although communication is necessary for distributed training, it may limit throughput scaling with increasing device count and cause compute resources to be idle when communication is on the critical path. Further, unlike a system's compute throughput, which accelerator designers have heavily focused on, network bandwidth has not scaled as much (e.g., 12× compute improvement versus 2× network bandwidth improvement [34]). If compute continues to scale more rapidly, when coupled with increasing communication volume, training future large-scale models on future systems will be inefficient. Thus, it is important to understand how Comp-vs.-Comm scale relative to one another as models scale and hardware evolves. To address

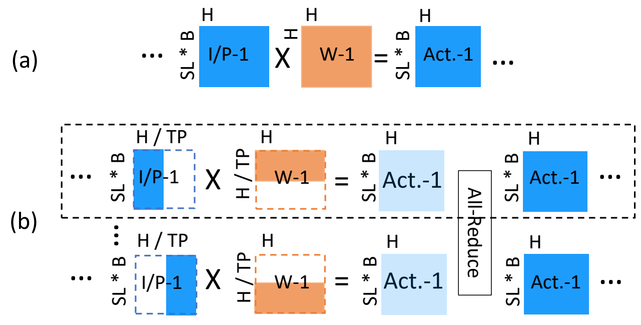


Figure 4: Layer operations (a) original, or w/ DP (b) w/ TP.

this we perform a multi-axial (algorithmic, empirical, hardware evolution) analysis of Comp-vs.-Comm scaling which will both inform and guide future system design to better support large-scale training of future models.

3. Comp-vs.-Comm: Algorithmic Analysis

An algorithmic analysis of compute and communication is important because it provides a strong foundation to draw meaningful conclusions about future models. Moreover, as we demonstrate (Section 4), it helps to create an empirical strategy to study model scaling on future hardware using existing hardware. Additionally, and equally importantly, it provides a system and infrastructure agnostic view of Comp-vs.-Comm scaling – ensuring that the takeaways are relevant regardless of studied system.

3.1. Distributed Techniques Studied

Although there are other distributed techniques and technique combinations, we study Transformers in the most commonly used data parallel (DP) and tensor parallel (TP) setups [15, 65]. DP and TP are imperative to divide and conquer large datasets and models, respectively. Furthermore, DP and TP are heavily supported in popular DNN frameworks such as TensorFlow and PyTorch.

3.2. Important Hyperparameters

The size, and thus cost, of model components is dictated by a model's hyperparameters [51]. As shown in Figure 4(a), the key hyperparameters that impact the size of weights, and activations in Transformers are: layer width or hidden dimension (H), input batch size (B), and input sequence length (SL). Although other hyperparameters are tuned during training (e.g., layer count, learning rate), they do not directly impact the size of operations.

Distributed setups can also impact the size of operations. In DP, since the model is replicated, operation size is unaffected. Conversely, as shown in Figure 4(b)'s dotted box, TP slices the operations. Hence, the number of devices a model is split across, the TP degree,[1] also impacts operation size. Thus, we use H, B, SL and TP to analyze Comp-vs.-Comm in an algorithmic and hardware-agnostic manner (Section 3.3).

1. We use TP to refer to both tensor parallelism and its degree.

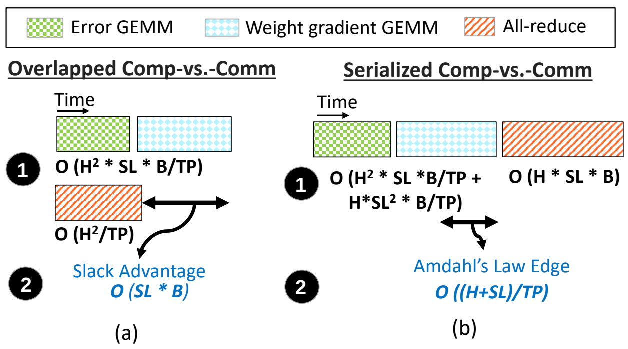


Figure 5: Comp's (a) slack over overlapped Comm. (b) edge over serialized Comm.

3.3. Amdahl's Law Edge for Compute

As described in Sections 2.3.2 and 2.3.3, a distributed setup with DP and TP introduces communication in the form of all-reduce. Here we consider the TP-related communication that all-reduces partial activations. As shown in Figure 5(b), this communication, hereafter referred to as *serialized communication*, is on the critical path of model execution. To assess compute's relative edge or Amdahl's Law edge (Section 2.3.3), we find the relative contribution of all compute and serialized communication operations in a Transformer block. Since a model consists of multiple Transformer blocks of the same size, studying compute and communication within a single block is sufficient to characterize this for an entire model. For compute, this includes matrix multiplications (GEMMs) which represent Transformer's key layers (attention and FC, Section 2.1) and other element-wise and reduction operations that constitute the remaining activation functions and normalization layers. However, modern Transformer implementations usually fuse [20, 23, 69, 75] non-GEMM operations with the preceding GEMM to increase on-chip data reuse.

Thus, our algorithmic analysis only considers the dominant GEMMs for compute. Since GEMMs are usually compute-bound, we evaluate their algorithmic cost as the number of operations (multiplies and adds) they perform: $2 \cdot M \cdot N \cdot K$ (where M, N, and K are matrix dimensions and are derived from model hyperparameters, as shown in Figure 4). In a Transformer block, there are three key sets of GEMMs [51], with computational complexities (with TP) shown in Equations 1- 4 (and in ① in Figure 5(b)).

$$\begin{aligned}\text{FC GEMM Ops.} &= 2 \cdot (4 \cdot H \cdot H/TP \cdot SL \cdot B) \\ &= \mathcal{O}(H^2 \cdot SL \cdot B/TP)\end{aligned} \tag{1}$$

$$\begin{aligned}\text{Attention GEMM Ops.} &= 2 \cdot (H/TP \cdot SL \cdot SL \cdot B) \\ &= \mathcal{O}(H \cdot SL^2 \cdot B/TP)\end{aligned} \tag{2}$$

$$\begin{aligned}\text{Linear GEMM Ops.} &= 3 \cdot 2 \cdot (H/TP \cdot H \cdot SL \cdot B) \\ &= \mathcal{O}(H^2 \cdot SL \cdot B/TP)\end{aligned} \tag{3}$$

$$\text{Total Comp. Ops.} = \mathcal{O}(H \cdot SL \cdot B/TP \cdot (H + SL)) \tag{4}$$

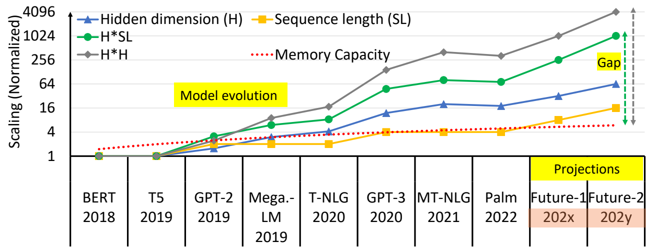


Figure 6: Model and device memory capacity trends.

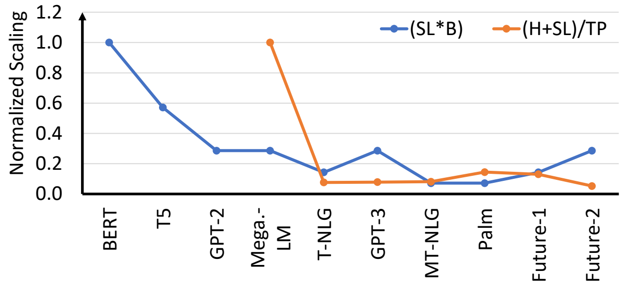


Figure 7: Algorithmic scaling of slack and edge.

For serialized communication, we consider the total bytes of data that are all-reduced. In TP, layers' output activations and errors are all-reduced. Their sizes are a multiple (depending on precision) of the GEMMs' output matrices sizes (i.e., $M \cdot N$), and can also be represented in terms of the hyperparameters (see Figure 4). In a Transformer block, there are four serialized all-reduce operations, all with complexity shown in Equation 5 (and in ① in Figure 5(b)).

$$\begin{aligned}\text{Total Comm. Bytes} &= (precision/8) \cdot (H \cdot SL \cdot B) \\ &= \mathcal{O}(H \cdot SL \cdot B)\end{aligned} \tag{5}$$

Using Equations 4 and 5, we find the ratio between the number of compute operations and communicated bytes in Equation 6. This represents the complexity of **Amdahl's Law edge** that compute has over communication (② in Figure 5(b)).

$$\begin{aligned}\textit{Amdahl's law edge} &= \mathcal{O}((H^2 \cdot SL \cdot B/TP)+ \\ &(H \cdot SL^2 \cdot B/TP))/\mathcal{O}(H \cdot SL \cdot B) \\ &= \boldsymbol{\mathcal{O}((H + SL)/TP)}\end{aligned} \tag{6}$$

This complexity has two implications. First, given the values of these hyperparameters in state-of-the-art Transformers (Section 3.5), with $(H + SL)$ being always greater than TP, compute (ops) enjoys an algorithmic edge over communication (bytes). Second, this edge can decrease if the required TP degree increases more than the increase in $(H + SL)$ between Transformer models, resulting in an overall increase in communication proportion.

3.4. Slack Advantage for Compute

Similar to our serialized communication analysis, we also study DP's *overlapped communication* that all-reduces partial gradients in backprop, as illustrated in Figure 5(a). We algorithmically analyze the relative cost of compute and overlapped communication and assess compute's ability to hide communication (Section 2.3.2). Unlike serialized communication, analyzing overlapped communication per Transformer layer is sufficient. Unlike TP, gradient communication occurs only in backpropagation, is done for every layer, and is usually overlapped with gradient/error calculating GEMMs. Thus, we can study the overlap efficacy by analyzing the compute and communication at every layer during backprop. For compute, we consider GEMMs which calculate backprop weight gradient (WG) and error (or input gradient); for communication we consider the WG size that is all-reduced. Absolute values of compute and communication can differ across layers: for example, with 4× layer widths, the compute and communication costs in FC layers are 4× those of attention. However, their complexities with respect to hyperparameters are the same. Thus, while Equations 7 and 8 derive these complexities for the FC layer (① in Figure 5(b)) they also hold and summarize algorithmic behavior for all Transformer layers.

$$\begin{aligned}\text{FC WG + Error GEMM Ops.} &= 4 \cdot (4 \cdot H \cdot H/TP \cdot SL \cdot B) \\ &= \mathcal{O}(H^2 \cdot SL \cdot B/TP)\end{aligned} \tag{7}$$

$$\begin{aligned}\text{Comm. bytes} &= (precision/8) \cdot (4 \cdot H \cdot H/TP) \\ &= \mathcal{O}(H^2/TP)\end{aligned} \tag{8}$$

$$\begin{aligned}\textit{Slack advantage} &= \mathcal{O}(H^2 \cdot SL \cdot B/TP)/\mathcal{O}(H^2/TP) \\ &= \boldsymbol{\mathcal{O}(SL \cdot B)}\end{aligned} \tag{9}$$

Equation 9 (② in Figure 5(b)) uses Equations 7 and 8 to find the ratio between the number of GEMM operations and total bytes communicated or all-reduced, and the complexity of compute's **slack** (i.e., ability to hide communication). This $SL \cdot B$ factor provides compute operations additional slack to hide the cost of bytes communicated. However, decreasing the input size $(SL \cdot B)$ can decrease the slack and potentially expose communication costs.

3.5. Model Scaling Stresses Compute Edge/Slack

Our analysis in Sections 3.4 and 3.3 show that, algorithmically, compute has both an Amdahl's Law edge over serialized communication and slack to hide the overlapped communication. However, the extent of this edge and slack varies depending on the hyperparameters: the edge grows as H or SL increases but decreases if TP increases. Similarly, slack grows with increasing B or SL. In recent years H and SL have increased considerably across Transformer models [12, 15, 18, 44, 53, 54, 65, 68].[2] As shown in Figure 6, these trends are expected to continue since larger

2. Although recent work has improved accuracy by increasing training token count instead of model size [30], scaling H is still the most widely used technique to improve model quality. Further, our empirical analysis shows that compute throughput scale faster than network bandwidth – thus increasing communication even for a fixed H.

H and SL are strongly correlated with improved model quality [54]. However, model parameters scale quadratically with H and activations scale linearly with both H and SL – thus increasing Transformers' memory requirements. Figure 6 uses $H \cdot H$ and $H \cdot SL$ values to show memory requirement scaling for parameters and activations. These results show that if the trend of linear scaling of device memory capacity continues, the gap between available device memory capacity and models' future memory demand will increase. Consequently, using smaller B's to reduce activation sizes, and larger TP slicing to distribute parameters have become imperative to limit memory pressure (detailed in Section 4.3.2 and Figure 9(b)). If this trend in B and TP exceeds the corresponding increase in H and SL, the resulting algorithmic edge ratios (i.e., $(H+SL)/TP$) and slack (i.e., $SL \cdot B$) can decrease, exposing additional communication on the critical path. We show this scaling in Figure 7, which plots compute's algorithmic slack and edge over communication for all studied Transformers, normalized to that of BERT's. Due to a considerable decrease in B (=1), compute's slack has reduced by $\sim$75%. Similarly, if TP is scaled to fit these models (details in Section 4.3.2) compute's edge can decrease by $\sim$80%. The $SL \cdot B$ values for futuristic models increase as we project larger SLs for them while B remains at the minimum of one. In reality, SL scaling is also often limited by memory capacity and will result in the slack being constant. Consequently, model scaling and memory capacity trends are stressing compute's algorithmic edge over communication.

This algorithmic analysis also does not account for the cost of executing an operation or communicating a byte. Thus, an individual Transformer's compute ops to communication bytes ratio does not directly translate to execution time ratio, and compute may actually have no/smaller edge and slack (explored further in Section 4). Nevertheless, our algorithmic analysis provides insights on how evolving Transformers affect edge and slack.

4. Comp-vs.-Comm: Empirical Analysis

Thus far our analysis has shown that compute enjoys an algorithmic edge over communication, but this edge is being stressed as models evolve (Section 3). We next use empirical analysis to quantify this edge. Since an exhaustive empirical study can be expensive, we propose a strategy based on our algorithmic analysis that uses existing hardware to project Comp-vs.-Comm for any future model on future hardware.

4.1. Empirical Analysis Challenges

An empirical analysis must be designed carefully because model evolution can cause an explosion of scenarios (hyperparameters) to consider and, consequently, experiments to run. This is further exacerbated when considering hardware evolution due to many hardware parameters. Thus, it is important to carefully identify variables of interest when designing experiments to study both model and hardware evolution. Moreover, even with a disciplined exploration of the hyperparameters and hardware parameters, profiling costs can still be very high, especially in scenarios requiring entire training iterations to be profiled. Thus, careful examination of the variable space, close attention to controlling profiling overheads, and high-fidelity model and hardware evolution designs are paramount to empirically study Comp-vs.-Comm scaling for future models and hardware.

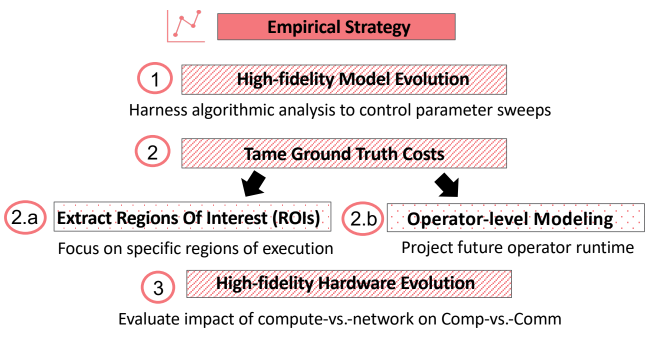

Empirical Strategy
1
High-fidelity Model Evolution
Harness algorithmic analysis to control parameter sweeps
2
Tame Ground Truth Costs
2.a
Extract Regions Of Interest (ROIs)
Focus on specific regions of execution
2.b
Operator-level Modeling
Project future operator runtime
3
High-fidelity Hardware Evolution
Evaluate impact of compute-vs.-network on Comp-vs.-Comm

Figure 8: Components of proposed empirical strategy.

4.2. Proposed Empirical Strategy

Next, we discuss the components (Figure 8) of our empirical strategy to overcome the aforementioned challenges.

4.2.1. High-fidelity Model Evolution. (Step ①)

To effectively study Comp-vs.-Comm scaling for future models, we must carefully consider model evolution. Section 3.5 demonstrated that historically models have scaled both the hidden dimension (H) and sequence length (SL) to improve accuracy. Other hyperparameters (batch-size B and degree of tensor-parallelism, TP) depend on a system's resources (e.g., compute and memory capacity). A naive but exhaustive exploration of such a hyperparameter space will help faithfully study model evolution. However, even after excluding unrealistic configurations (e.g., large H and large B with a small TP), it would require running an impractically large number of experiments.

We overcome this challenge by anchoring on our algorithmic analysis. Specifically, we use the Comp-vs.-Comm scaling ratios identified in Section 3 to design controlled experiments. For scenarios where communication is overlapped with computation (e.g., DP), since algorithmically the Comp-vs.-Comm ratio is $\mathcal{O}(SL \cdot B)$, we focus on sweeping $SL \cdot B$ for different H values to study how compute's slack advantage scales for future models. However, this still requires several different (from H, and $SL \cdot B$) iterations to be profiled. Furthermore, for serialized communication (e.g., TP), since the ratio of Comp-vs.-Comm is $\mathcal{O}((H+SL)/TP)$, we can only factor out B. We identify additional strategies to tame this exploration (Section 4.2.2).

4.2.2. Taming Ground-truth Cost (Step. ②)

Although algorithmic analysis helps prune the search space, further solutions are needed to reduce profiling costs. Accordingly, similar to prior work [52] we use application understanding to identify and study only specific fractions of executions where possible. When entire iteration times are required, we rely on high-fidelity operator-level models

Parameter / Setup	Value
H	1K, 2K, 4K, 8K, 16K, 32K, 64K
{B}, {SL}	{1,4}, {1K, 2K, 4K, 8K}
{TP degree}, {DP degree}	{ 4, 8, 16, 32, 64, 128, 256}, {Any}

TABLE 1: Parameters and setup of models studied.

to project the runtime of different Transformer configurations without actually running them. We further explain these strategies below.

Region of Interest (ROI) Extraction (Step 2a): As discussed in Section 3.4, to study overlapped Comp-vs.-Comm (e.g., in DP) it suffices to extract the specific compute (e.g., GEMMs) and communication fractions (e.g., All-reduce) which will manifest for future models and profile the execution of only these regions in hardware. These controlled experiments help us study how compute's slack, to hide communication, will evolve as models scale and hardware evolves, and avoids the cost of running the entire training iteration for all configurations of interest.

Operator-level Models (Step 2b): For serialized Comp-vs.-Comm (e.g., in TP), executing ROI regions is insufficient. To quantify how much Amdahl's Law edge compute enjoys over communication, it is necessary to study entire training iterations. However, we observe that building high-fidelity operator-level models and combining their results can help us project entire network behavior. Specifically, for every operator in the Transformer layer's execution that repeats during a training iteration (e.g., GEMMs and layer-normalization), we use algorithmic analysis to identify hyperparameters that affect its execution time. Given the operator's execution time for a hyperparameter configuration, we can project the execution time for any different set of hyperparameter values. Thus, these operator-level models project Transformer behavior for a wide variety of hyperparameter values without significant profiling costs. Moreover, our evaluation (Section 4.3.8) shows that these operator-level models are reliable and accurately capture the behavior of operations with changing hyperparameters.

4.2.3. High-fidelity Hardware Evolution (Step. 3)

Similar to model evolution, a disciplined hardware parameter search space is equally important. Accordingly, we identified the key drivers important to Comp-vs.-Comm scaling: compute throughput (FLOPS), network bandwidth, and memory bandwidth. Of these, we focus on the first two factors. Although communication performance is impacted by all three factors, efficient communication primitive (e.g., all-reduce) implementations are pipelined. Thus, they can overlap memory accesses with network transfers – and since network transfers usually dominate, memory bandwidth has a relatively smaller impact. Moreover, while compute operations depend on both compute FLOPS and memory bandwidth, key Transformer operations (e.g., GEMMs) are often compute-bound (e.g., Gshard reports >85% peak FLOPS utilization [37]) and have low memory bandwidth utilization [51]. Thus, we focus on compute FLOPS and network bandwidth, specifically on their *relative scaling*

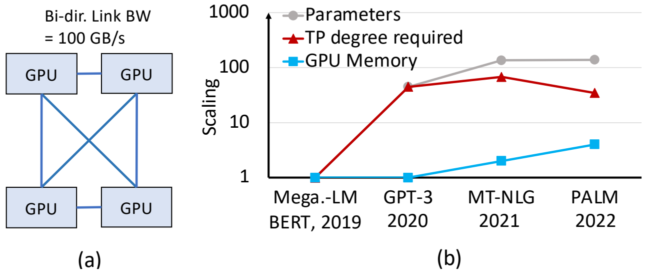


Figure 9: System: (a) 4-GPU node (b) TP scaling with model size.

ratios based on historical data for GPUs from different vendors (discussed in Section 4.3.6).

4.2.4. Benefits Compared to Exhaustive Profiling.

Our empirical strategy reduces execution and profiling costs (Section 4.3.8) of Comp-vs.-Comm analysis for future models on future hardware. First, our algorithmic analysis helps identify a subset of hyperparameters to sweep, limiting the combinations to consider ($SL \cdot B$ rather than individually sweeping SL and B). Second, the operator-level models enable projection of iteration times for many (196) different configurations using the execution and profiling of a single iteration. Finally, focusing on specific ROIs avoids executing end-to-end iterations for overlapped communication.

4.3. Observations from Experimental Analysis

4.3.1. System Setup.

We run experiments based on our empirical strategy using a system with an AMD Ryzen™ Threadripper™ CPU and four AMD Instinct™ MI210 accelerators (GPUs) [8] (Figure 9(a)), each with 64GB HBM. The GPUs are fully connected using AMD Infinity Fabric™ links with bidirectional link bandwidth of 100GB/s. These links form multiple rings, providing a peak ring all-reduce bandwidth of 150GB/s. We also calibrated our system's performance and found it was similar to prior work using other commercial systems [32]. Finally, our software stack uses AMD's open source ROCm™ version 5.2 [6] with PyTorch v1.7, the rocBLAS [5] BLAS library, and the RCCL [3] communication collectives library.

4.3.2. Models & Cluster Setup.

To study a range of Transformers (Figure 6) we evaluate the hyperparameter combinations and distributed setups listed in Table 1.

Model Setup (*H*, *B*, *SL*): Scaling Transformers typically involves scaling H and SL [54]. Thus, for H we examine power-of-two values up to 16K and for SL up to 2K, as they represent a wide spectrum of modern Transformers. Additionally, to project future model behavior we scale H to 32K (Future-1 with 1 trillion parameters) and 64K (Future-2 with ten trillion parameters) with SLs of 4K and 8K, respectively. For B, we consider small values of one and four. Smaller Bs (and larger TPs discussed in Section 3.5) are required to bridge the large gap between required and available memory capacity (Section 3.5). In fact, most modern larger models (e.g., MT-NLG [68] and PALM [15]) already use the smallest B of one.

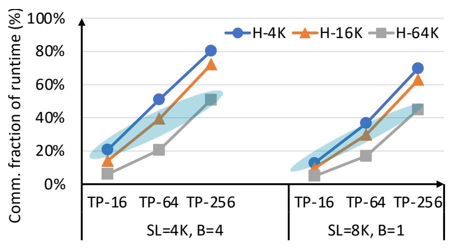


Figure 10: Fraction of serialized comm. time.

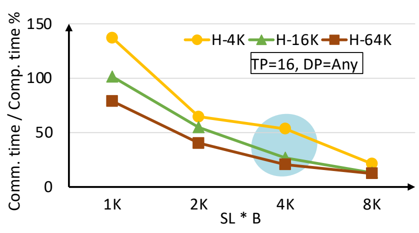


Figure 11: Overlapped comm. as a percentage of comp. time.

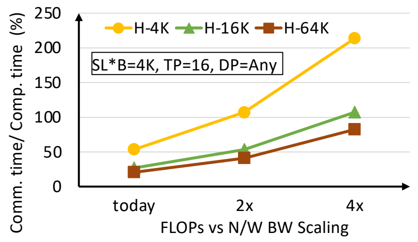


Figure 12: Hardware evolution impact on overlapped comm. as a percentage of comp. time.

Training Cluster Setup (TP degree, DP degree): We study a range of *TP* and *DP* degrees:

TP degree: We determine the appropriate *TP* range based on modern Transformer setups. We start with one of the largest, 3.9B parameters, Megatron-LM models (Mega.-LM_BERT), the first publicly known Transformer to use tensor-parallelism with *TP* of eight. To estimate the *TP* for a future Transformer, we consider device memory capacity and model size. Assuming a capacity of eight devices (=*base_TP*) is required for Mega.-LM_BERT, we estimate a larger model's *TP* by scaling up *base_TP* by the ratio (p) of its model size compared to that of Mega.-LM_BERT. To account for potential device memory capacity increases in the same time period, we divide the projected *TP* by the memory capacity scaling ratio (s) in that time period. Thus, the required TP degree is $base\_TP * (p/s)$. Figure 9(b) shows the scaling of Transformers, and device memory capacities, as well as the resulting scaling of *TP* (p/s) required to fit the Transformers, all normalized to Mega.-LM_BERT. Since memory scaling (16GB [2] to 64GB [8]) has not been proportional to Transformer model scaling (8.3B [65] to 540B [15]), *TP* needs to be scaled by 40-60$\times$, leading to a required *TP* degree of ($*8$) $\sim$250-550. Although *TP* has increased over the last few years as models scale, considerable innovations in interconnect technology will be necessary to realize such large *TP*s. Furthermore, pipeline parallelism can also be relied on to limit *TP*. Consequently, we study a range of *TP* values up until 256.

DP degree: Our data-parallel empirical analysis is largely agnostic to *DP* degree. Unlike TP, compute FLOPS and overall communication size are not dictated by *DP* degree. Furthermore, while we use a four-GPU ($N = 4$) setup, it also provides us with a reasonable, albeit conservative, estimate of communication time on larger setups because (ring) all-reduce traffic scaling is small at large device counts, which scales with (($N-1$)/N $\sim$1 for large N). However, increasing device count also increases synchronization cost between devices, causing the actual communication time to be slightly higher. Furthermore, DP training is usually setup on large-scale multi-node, often heterogeneous, systems with slower inter-node links. Since we did not have access to any of these machines, we instead optimistically estimate the communication times using intra-node links and discuss the implications of this in Section 4.3.7.

4.3.3. Profiling Setup.

For the overlapped communication analysis, as discussed in Section 4.2.2 we extract relevant regions from training iteration (compute and communication operations) and execute only these relevant regions for all possible hyperparameter combinations under consideration (Table 1). Although in reality they execute concurrently, we execute and study them in isolation to avoid interference slowdowns from shared resources and to understand their optimal characteristics in isolation. For serialized communication analysis, we first profile BERT [18] training iterations on a single GPU as a baseline. Next, we employ our operator-level models (Section 4.2.2) to project training runtime for hundreds of Transformer configurations. Finally, we use rocProf [4] to measure GPU kernel execution times.

4.3.4. Amdahl's Law Edge Analysis.

Using our empirical strategy (Section 4.2), hyperparameter trends (Section 3.5), and estimated TP values (Section 4.3.2), we project the proportion of serialized communication as compared to compute. Figure 10 shows the fraction of Transformer training time spent on communication for a subset of varying H, SL, and TP values. It includes a medium Transformer ($\sim$T-NLG [44]), one of today's largest Transformer models ($\sim$PALM [15]), and a futuristic Transformer (Future-2). For a fixed H and $SL \cdot B$ (a line), the communication proportion increases with increasing TP. Conversely, with fixed TP it decreases with either an increasing H or SL. These trends mirror our algorithmic takeaways (Section 3.3). Furthermore, the communication fraction is considerable and increases as models scale. Models of different sizes require different TP values to train (discussed in Section 4.3.2). While a TP degree of 16 can potentially suffice for a model with $H = 4K$ (e.g., T-NLG), it has to be scaled for larger models (e.g., TP of 64 for H of 16K). These parameter combinations are highlighted in blue in Figure 10 and show that communication increases to a considerable 50% of the execution time for a model with $H = 64K$ (Future-2). This trend also correlates with our algorithmic takeaways (Section 3.3): with SL constant, and similar scaling of H and TP, the denominator of $(H+SL)/TP$ scales much more, causing compute's Amdahl's Law edge to decrease.

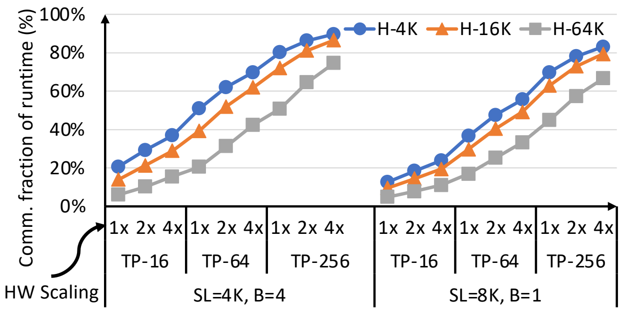


Figure 13: Impact of hardware evolution on fraction of serialized communication time.

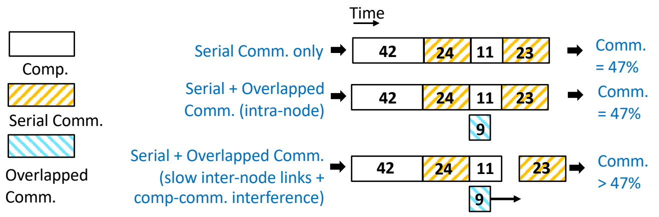

Time
Comp.
Serial Comm.
Overlapped Comm.
Serial Comm. only
Serial + Overlapped Comm. (intra-node)
Serial + Overlapped Comm. (slow inter-node links + comp-comm. interference)
42
24
11
23
9
Comm. = 47%
Comm. = 47%
Comm. > 47%

Figure 14: Overall Comp-vs.-Comm Case Study. Setup: H=64K, B=1, SL=4K, TP degree=128, flop-vs.-bw scale=4x.

4.3.5. Slack Advantage Analysis. We estimate the fraction of time that communication is overlapped with compute using our empirical strategy (Section 4.2) and hyperparameter trends (Section 3.5). This helps estimate both how well compute's slack advantage can hide communication costs and how this slack scales. Moreover, these estimates hold irrespective of the degree of *DP*. Figure 11 shows that the overlapped time decreases as the product of *SL* and *B* ($SL \cdot B$ in Figure 11) increases, similar to our algorithmic takeaway in Section 3.4. Additionally, the overlap percentage is higher at smaller *H*, causing smaller remaining slack. Our algorithmic analysis did not account for this since it is an artifact of hardware execution. Additionally smaller *H*, and thus smaller communication sizes do not fully use the network bandwidth capacity of devices that larger sizes can. This results in a sub-linear increase in communication costs until the network bandwidth saturates. Conversely, compute operations are large enough to saturate compute FLOPS. Thus, the slower communication at smaller sizes creates a larger overlap and leaves less compute slack.

Furthermore, the communication overlap percentages are very high, ranging from 17% to 140% for the range of H, SL, and *B* values, with a fixed *TP* degree of 16 and irrespective of the *DP* degree. In particular, the highlighted blue region shows that for the common $SL \cdot B$ value of 4K, across a range of models, communication forms 20-55% of compute time, leaving smaller compute slack. Moreover, this percentage is likely to be much higher as this communication usually occurs in large multi-node setups with slower network links than the high-bandwidth intra-node links we study.

4.3.6. Future Hardware Analysis. Thus far we have estimated the Comp-vs.-Comm costs while training Transformers on current systems. However, evolving hardware can change these estimates and shift application bottlenecks. Thus, we next estimate the Comp-vs.-Comm costs for future systems, using past hardware trends to help inform future system design. First we estimate the relative scaling of compute FLOPS versus network bandwidth, which we call *flop-vs.-bw*. This value varies across GPU generations as well as vendors. Between 2018 and 2020, compute FLOPS scaled by $\sim$5× [46, 48] and $\sim$7× [2, 7], while corresponding network bandwidth scaled only by $\sim$2× [46, 48] and $\sim$1.7× [2, 7], respectively. This implies that compute FLOPS have scaled relatively more than network bandwidth: by $\sim 2-4\times$. We use these relative *flop-vs.-bw* ratios to scale the compute time estimated in Sections 4.3.4 and 4.3.5 and project its resulting slack advantage and Amdahl's Law edge over communication. Reducing precision can further disproportionately scale compute FLOPS more than network, causing this ratio to be much higher (discussed further in Section 6, *Number-formats*).

Figure 13 shows that, with 2× and 4× flop-vs.-bw scaling, serialized communication starts to dominate training execution. The range increases from 20-50% to 30-65% and 40-75%, respectively, for the configurations in Section 4.3.4. Similarly, compute acceleration also reduces, or even eliminates, compute's slack to overlap communication. Figure 12 shows that the overlapped communication is 50-100% and 80-210% of compute time with 2× and 4× flop-vs.-bw scaling, and communication is exposed (i.e., on the critical path) in many cases (when $>= 100\%$). Furthermore, these communication percentages will increase in inter-node setups (discussed in Section 4.3.7). Thus, if similar trends in hardware evolution continue, communication will become a critical bottleneck in training Transformers.

4.3.7. End-to-end Comp-vs.-Comm Case Study: Combining Serialized & Overlapped Communication. Figure 14 shows the combined impact of both TP and DP for a large futuristic Transformer model. 47% of time is spent on serialized communication while 9% is spent on overlapped communication. Since the latter is completely hidden by independent (backprop GEMM) computations, 47% of the overall communication is on the critical path.

Lower inter-node communication bandwidth and interference slowdown also affect overlapped communication ($\sim$8× [57]). The former is pertinent since portions of DP's overlapped communication may be sent over inter-node links. The latter is pertinent since communication can potentially slow down due to interference during its concurrent execution with compute [57]. The third scenario in Figure 14 shows their impact – DP-directed communication is no longer completely hidden. Thus, with TP-directed communication serialized and DP-directed communication only partially overlapped, total communication will become a larger bottleneck for future Transformer training.

4.3.8. Evaluating Operator-level Model. Hardware execution of all models and system setups can provide a more accurate Comp-vs.-Comm analysis. Although ROI

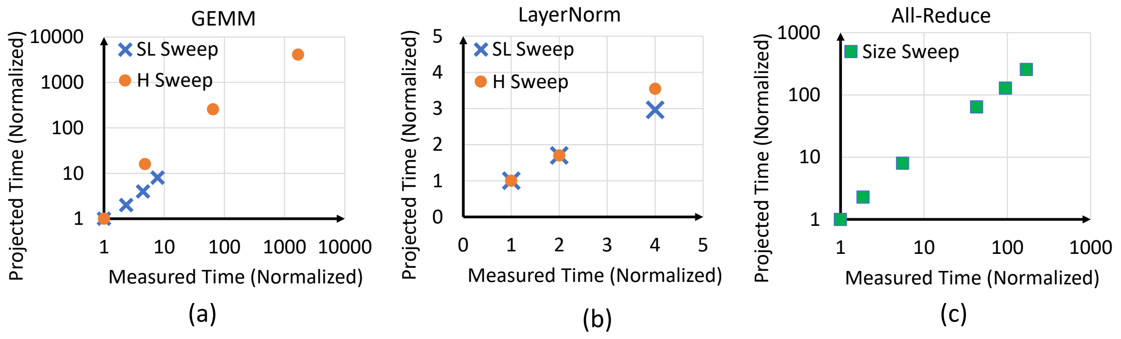


Figure 15: Effectiveness of Operator-level modeling.

extraction makes this possible for the overlapped communication study in DP, measuring the end-to-end model breakdown in order to evaluate serialized communication for all configurations is impractical. Thus, to study the end-to-end breakdown of a future model's training execution we project the runtime of all its individual components using our operator model (Section 4.2.2). Here we evaluate the approach's effectiveness and benefits:

Accuracy: To evaluate the operator-level model's effectiveness, we compare the projected runtimes of operations and communication against those measured on hardware while sweeping hyperparameters and data size, respectively. Figure 15 shows this comparison for three operators which cover a spectrum of Transformer hardware characteristics: compute-bound GEMMs, memory-bandwidth bound LayerNorm, and the all-reduce communication collective. For GEMM and LayerNorm, we show the projected and measured execution times for different hyperparameter (H, SL) values. For all-reduce, we compare them for a range of array sizes. These results are normalized by the measured execution time of the operation (or communication) using the base hyperparameter (or array size) used for projections.

Figure 15(a) shows how linearly scaling Transformer GEMMs' runtimes with SL accurately captures their hardware behavior as SL varies. Similarly, scaling GEMM's runtime quadratically with the H (layer width) captures hardware trends with increasing H. However, individual errors in projecting GEMM runtimes are not negligible as operator efficiency changes with input size. Generally, GEMM efficiency improves as input sizes increase, causing runtime scaling ratios to be smaller than the ratios of input sizes. This pattern continues until GEMMs achieve near-peak efficiency, after which runtime scaling becomes analogous to input size scaling. Thus, errors with projecting future GEMM runtimes using a small operation size as the baseline can be large. Although we use BERT as the baseline model, its GEMMs do not achieve peak efficiency. This results in the projected GEMM runtimes for future models to be higher, as demonstrated in Figure 15(a). Although the smaller-than-projected GEMM runtimes suggest that the proportion of TP-related communication for larger models is slightly higher, it does not alter our main insights. For example, while the error may improve by using a larger baseline model, and thus operation sizes, Figure 15's trends and our key takeaways will still hold. Overall, across all studied GEMMs, the model projects runtime with an error of $\sim 15\%$. Similarly, Figure 15(b) shows we accurately model LayerNorm's runtime, which is linear with both SL and H: $\sim 7\%$ geomean error. Finally, Figure 15(c) shows our model accurately models all-reduce trends as data size varies: $\sim 11\%$ geomean error. Although efficiency also impacts the projections of these operators (e.g., due to better memory and network bandwidth utilization), its impact is small since they are usually close to saturation in the baseline models.

Profiling Speedups: Finally, exhaustively studying hundreds of configurations (parameter combinations from Table 1) without actually executing them saves considerable profiling time and effort. Specifically, our strategy avoids executing $\sim$198 different Transformers (some very expensive), reducing profiling costs by three orders of magnitude (2100×) compared to serialized Comp-vs.-Comm for the 198 configurations. We also avoid executing end-to-end iterations, specifically the forward propagation, to estimate the overlapped Comp-vs.-Comm costs. This speeds up profiling by 1.5×.

5. ML/System Evolution Recommendations

Our Comp-vs.-Comm analysis demonstrates that communication is starting to become a considerable bottleneck for distributed training. Here we discuss some promising techniques that stand to tackle this challenge and also discuss how our analysis influences their potential improvements.

System-aware ML Evolution: Design of novel DNNs is often influenced by constraints of underlying hardware and vice-versa (e.g., number formats in ML).

Comp-vs.-Comm influence: Our analysis shows that stressing certain hyper-parameters more than others (e.g., scale SL more than H) stands to strengthen compute vis-a-vis communication. This is so, as first, scaling SL improves both the edge and the slack compute has over communication (Section 3). While scaling H helps increase compute's edge, it stresses memory capacity (for parameters) quadratically. As such, model evolution which scales SL more than H is likely to have a lower communication fraction than vice-a-versa. Note, existing works which scale SL have shown interesting results [13, 14].

Communication Offloads/Fusion: Some techniques offload communication from an accelerator (e.g., GPU) to a co-processor (e.g., ASIC, FPGA, DPU) [9, 57] which are

specialized to accelerate communication. They can address communication which can be overlapped with computation (e.g., data parallelism). To tackle communication on critical path, techniques that break communication abstractions and optimize for pipelining/overlap of data generation and communication can be employed [22, 32, 76].
Comp-vs.-Comm influence: Our analysis indicates that both serialized and overlapped communication are important. Consequently, a judicious combination of both offload and fusion will be necessary for future Transformers.

Processing-in-memory (PIM): Several commercial realizations of Processing-in-memory (PIM), which push compute units closer to memory, have recently emerged [63, 67].
Comp-vs.-Comm influence: Lowering communication-induced memory traffic can help improve efficiency. This can be enabled by efficient support for in-memory atomics with PIM which can lower memory traffic required for the reduction computation in an all-reduce primitive. This also stands to lower interference in memory between communication and computation executing on the accelerator.

Processing-in-network (PIN): Processing data during traversal is also promising [61, 64]. Specifically, techniques that enhance existing network switches to execute collectives [25, 34, 40] halve the network's transmitted bytes compared to a bandwidth-optimal ring all-reduce [10]. This is because devices only send their copies of data to the switches once and receive the reduced version from the switches. Unlike in software-based ring/direct all-reduce approaches where devices send and receive arrays twice; for reduce-scatter and all-gather. However, PIN-based techniques are limited to topologies with switches.
Comp-vs.-Comm influence: As switch-based collectives are limited in their bandwidth benefit ($\sim$2x), our analysis shows that judiciously combining them with fusion will be necessary for future Transformers given the fraction of execution time bottlenecked by communication.

6. Discussion

Beyond DP & TP: While we focus on DP and TP, communication from other distributed techniques can be folded into our analysis. *Mixture-of-experts (MoE)* sparsely activate parts of a network to reduce computational costs [21, 55]. Besides DP and TP, MoEs also deploy expert parallelism with additional serialized all-to-all communication – which can be incorporated into our serialized communication analysis. Overall, due to this additional communication and reduced computation, MoEs potentially increase the fraction of communication even further. *Pipeline Parallelism (PP)* partitions a model to assign a subset of layers to each device such that devices execute their layers in a pipelined manner [31]. We do not focus on PP as it adds pipeline bubbles which either degrades efficiency or requires a large number of micro-batches, which add memory pressure and degrade model quality. Nevertheless, our overlapped communication methodology/analysis can be extended to include the peer-to-peer communication of activations between PP devices. Finally, the Fully Shared Data Parallel (FSDP) technique combines DP's and TP's benefits by distributing weights amongst DP devices and asynchronously gathering them just before layer computation. This adds additional overlapped communication in both forward and backward execution passes, which we could extend our slack analysis to examine.

Large/Other System Setups: LLM training usually uses both DP and TP: TP is employed within nodes and DP across nodes. Thus, DP-related communication occurs over slower inter-node links (e.g., Ethernet). While our empirical analysis focuses on intra-node setups with faster network links, it can be extended to encompass other network types. Nonetheless, our algorithmic analysis for identifying slack remains applicable and can also be utilized to reduce the time and effort needed for empirically estimating slack/overlap in large clusters.

Finally, while our empirical analysis uses a single GPU/hardware type, it can be extended to consider other hardware types (e.g., accelerators or GPU systems from other vendors). Recent work has shown that Transformers operations runtime can be calculated using their sizes and hardware specifications such as FLOPS, memory bandwidth, and intra-node bandwidth (albeit with efficiency considerations) [45]. Thus, our operator-level model could also be enhanced to project runtimes for another device using the ratios of the devices' specifications.

Large System Memory: Techniques to place the model state in system memory (CPU-attached DDR, NVMe memory) can help reduce accelerator memory pressure [49, 56, 60]. While this limits the required model-parallel (TP or PP) degrees and inter-accelerator communication, it can increase training time due to the limited compute capacity of fewer devices. Nevertheless, our methodology can be used to model communication for the resulting TP and be extended to include additional overlapped communication between CPU/NVMe and accelerator memory.

Number-formats: Number formats with lower number of bits [43, 62] have both computational and communication benefits during training. Our analysis and methodology, although for state-of-art mixed-precision training, are largely agnostic to the formats. Further, compute time can potentially decrease more than communication at smaller number formats. As such, the key takeaways of our analysis will likely carry over to these alternate formats.

Fine-tuning & Inference: Our takeaways hold for fine-tuning since it uses the same techniques and model as pre-training. Conversely, inference has much smaller memory requirements [24, 29] and thus can avoid distributed setups and communication. If deployed in distributed setups [50, 55], our proposal and takeaways will continue to hold.

Other DNNs: While we focus on Transformers due to their generality, our proposed methodology can be translated as is and/or extended to other DNNs. Specifically, our insights on ROI extraction and operator-level projections can be easily translated to other models. Similarly, an algorithmic analysis-based empirical strategy, as proposed by our work, can be extended for other DNNs.

7. Related work

DNN Characterization DNNs, especially Transformers, are an important application domain that are driving system optimizations. Consequently, there have been several works on benchmarking and characterizing them [1, 26, 42, 58, 74, 79–81, 81]. However, unlike our work, these focus on compute bottlenecks in single-device DNN executions and thus do not characterize the communication costs that arise in multi-device, distributed setups. Instead, we focus on characterizing the relative cost of communication compared to compute operations and show that more communication-focused innovations will be needed in the future.

Studying & Accelerating Communication: Other works study and/or optimize for communication in distributed setups [16, 34, 51, 57, 76]. However, unlike our work they do not examine how communication costs, and thus benefits of their optimizations, evolve across different Transformers, hardware capability, and different distributed techniques. While some works [19, 47] examine the throughput impact of sweeping a subset of hyperparameters, they either do not include in-depth characterization that examines the behavior or are on a single device.

8. Conclusion

Scaling of Transformers models and their datasets has necessitated very large-scale distributed setups, which raises the key question: *how will compute vs. communication (**Comp-vs.-Comm**) scale as models scale and hardware evolves?* We conduct a multi-axial (algorithmic, experimental, hardware evolution) analysis of Comp-vs.-Comm scaling for Transformer models. Our system-agnostic, algorithmic analysis highlights that while compute has enjoyed an edge over communication, future model and hardware trends are likely to make communication dominant soon. We also empirically study Comp-vs.-Comm for future Transformer models as hardware evolves. By extracting specific regions of interest and modeling future operator runtimes, we enable the study of hundreds of future Transformers/hardware scenarios with 2100× less profiling costs. These experiments validate that communication will play an increasingly large role (40-75%) in a distributed training setup as models scale. Overall, our multi-axial analysis shows the need for effective scaling of communication capabilities, and we conclude with a discussion of how our analysis influences some promising techniques and technologies.

References

[1] R. Adolf, S. Rama, B. Reagen, G.-y. Wei, and D. Brooks, "Fathom: Reference Workloads for Modern Deep Learning Methods," in *IEEE International Symposium on Workload Characterization*, ser. IISWC. Washington, DC, USA: IEEE, Sept 2016, pp. 1–10.

[2] AMD, "AMD INSTINCT™ MI50 ACCELERATOR," https://www.amd.com/en/products/professional-graphics/instinct-mi50, 2018.

[3] ——, "AMD's ROCm Communication Collectives Library," "https://github.com/ROCmSoftwarePlatform/rccl/wiki", 2018.

[4] AMD, "AMD ROCm Profiler," "https://rocmdocs.amd.com/en/latest/ROCm_Tools/ROCm-Tools.html", 2019.

[5] AMD, "AMD's BLAS Library," "https://github.com/ROCmSoftwarePlatform/rocBLAS", 2019.

[6] ——, "ROCm, a New Era in Open GPU Computing," "https://rocm.github.io/", 2019.

[7] ——, "AMD Instinct™ MI100 Accelerator," "https://www.amd.com/en/products/server\-accelerators/instinct\-mi100", 2020.

[8] ——, "AMD INSTINCT™ MI210 ACCELERATOR," https://www.amd.com/en/products/server-accelerators/amd-instinct-mi210, 2022.

[9] ——, "Distributed Services Card (DSC)," https://www.amd.com/system/files/documents/pensando-dsc-200-product-brief.pdf, 2023.

[10] Baidu, "Baidu all-reduce," https://github.com/baidu-research/baidu-allreduce, 2017.

[11] N. Benaich and I. Hogarth, "State of ai report 2022," https://www.stateof.ai/, 2022.

[12] T. Brown, B. Mann, N. Ryder, M. Subbiah, J. D. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, S. Agarwal, A. Herbert-Voss, G. Krueger, T. Henighan, R. Child, A. Ramesh, D. Ziegler, J. Wu, C. Winter, C. Hesse, M. Chen, E. Sigler, M. Litwin, S. Gray, B. Chess, J. Clark, C. Berner, S. McCandlish, A. Radford, I. Sutskever, and D. Amodei, "Language Models are Few-Shot Learners," in *Advances in Neural Information Processing Systems*, ser. NeurIPS, H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin, Eds., vol. 33. Red Hook, NY, USA: Curran Associates, Inc., 2020, pp. 1877–1901. [Online]. Available: https://proceedings.neurips.cc/paper/2020/file/1457c0d6bfcb4967418bfb8ac142f64a-Paper.pdf

[13] Cerebas, "Context is Everything: Why Maximum Sequence Length Matters," https://www.cerebras.net/chip/context-is-everything-why-maximum-sequence-length-matters/, 2022.

[14] Cerebras, "Genomics in Unparalleled Resolution: Cerebras Wafer-Scale Cluster Trains Large Language Models on the Full COVID Genome Sequence," https://www.cerebras.net/blog/genomics-in-unparalleled-resolution-cerebras-wafer-scale-cluster-trains-large-language-models-on-the-full-covid-genome-sequence, 2022.

[15] A. Chowdhery, S. Narang, J. Devlin, M. Bosma, G. Mishra, A. Roberts, P. Barham, H. W. Chung, C. Sutton, S. Gehrmann *et al.*, "Palm: Scaling language modeling with pathways," *arXiv preprint arXiv:2204.02311*, 2022.

[16] M. Cowan, S. Maleki, M. Musuvathi, O. Saarikivi, and Y. Xiong, "Msc-clang: Microsoft collective communication language," in *Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2*, 2023, pp. 502–514.

[17] Z. Dai, Z. Yang, Y. Yang, J. Carbonell, Q. V. Le, and R. Salakhutdinov, "Transformer-XL: Attentive Language Models Beyond a Fixed-Length Context," *CoRR*, vol. 1901.02860, 2019.

[18] J. Devlin, M. Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding," in *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, ser. NAACL-HLT, J. Burstein, C. Doran, and T. Solorio, Eds. Association for Computational Linguistics, 2019, pp. 4171–4186. [Online]. Available: https://doi.org/10.18653/v1/n19-1423

[19] D. Dice and A. Kogan, "Optimizing Inference Performance of Transformers on CPUs," *CoRR*, vol. abs/2102.06621, 2021.

[20] I. El Hajj, J. Gomez-Luna, C. Li, L.-W. Chang, D. Milojicic, and W.-m. Hwu, "KLAP: Kernel launch aggregation and promotion for optimizing dynamic parallelism," in *49th Annual IEEE/ACM International Symposium on Microarchitecture*, ser. MICRO, 2016, pp. 1–12.

[21] W. Fedus, B. Zoph, and N. Shazeer, "Switch transformers: Scaling to trillion parameter models with simple and efficient sparsity," *CoRR*, vol. abs/2101.03961, 2021. [Online]. Available: https://arxiv.org/abs/2101.03961

[22] Y. Feng, M. Xie, Z. Tian, S. Wang, Y. Lu, and J. Shu, "Mobius: Fine tuning large-scale models on commodity gpu servers," in *Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2*, 2023, pp. 489–501.

[23] J. Fousek, J. Filipovič, and M. Madzin, "Automatic Fusions of CUDA-GPU Kernels for Parallel Map," *SIGARCH Comput. Archit. News*, p. 98–99, Dec. 2011.

[24] A. Gholami, S. Kim, Z. Dong, Z. Yao, M. W. Mahoney, and K. Keutzer, "A survey of quantization methods for efficient neural network inference," 2021. [Online]. Available: https://arxiv.org/abs/2103.13630

[25] R. L. Graham, D. Bureddy, P. Lui, H. Rosenstock, G. Shainer, G. Bloch, D. Goldenerg, M. Dubman, S. Kotchubievsky, V. Koushnir, L. Levi, A. Margolin, T. Ronen, A. Shpiner, O. Wertheim, and E. Zahavi, "Scalable hierarchical aggregation protocol (sharp): A hardware architecture for efficient data reduction," in *2016 First International Workshop on Communication Optimizations in HPC (COMHPC)*, 2016, pp. 1–10.

[26] U. Gupta, S. Hsia, V. Saraph, X. Wang, B. Reagen, G.-Y. Wei, H.-H. S. Lee, D. Brooks, and C.-J. Wu, "DeepRecSys: A System for Optimizing End-To-End At-Scale Neural Recommendation Inference," in *ACM/IEEE 47th Annual International Symposium on Computer Architecture*, ser. ISCA, 2020, pp. 982–995.

[27] H. Hassan Awadalla, A. Aue, C. Chen, V. Chowdhary, J. Clark, C. Federmann, X. Huang, M. Junczys-Dowmunt, W. Lewis, M. Li, S. Liu, T.-Y. Liu, R. Luo, A. Menezes, T. Qin, F. Seide, X. Tan, F. Tian, L. Wu, S. Wu, Y. Xia, D. Zhang, Z. Zhang, and M. Zhou, "Achieving Human Parity on Automatic Chinese to English News Translation," March 2018.

[28] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016, pp. 770–778.

[29] G. Hinton, O. Vinyals, and J. Dean, "Distilling the knowledge in a neural network," 2015. [Online]. Available: https://arxiv.org/abs/1503.02531

[30] J. Hoffmann, S. Borgeaud, A. Mensch, E. Buchatskaya, T. Cai, E. Rutherford, D. de Las Casas, L. A. Hendricks, J. Welbl, A. Clark, T. Hennigan, E. Noland, K. Millican, G. van den Driessche, B. Damoc, A. Guy, S. Osindero, K. Simonyan, E. Elsen, J. W. Rae, O. Vinyals, and L. Sifre, "Training compute-optimal large language models," 2022.

[31] Y. Huang, Y. Cheng, A. Bapna, O. Firat, M. X. Chen, D. Chen, H. Lee, J. Ngiam, Q. V. Le, Y. Wu, and Z. Chen, *GPipe: Efficient Training of Giant Neural Networks Using Pipeline Parallelism.* Red Hook, NY, USA: Curran Associates Inc., 2019.

[32] A. Jangda, J. Huang, G. Liu, A. H. N. Sabet, S. Maleki, Y. Miao, M. Musuvathi, T. Mytkowicz, and O. Saarikivi, "Breaking the computation and communication abstraction barrier in distributed machine learning workloads," in *Proceedings of the 27th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*, 2022, pp. 402–416.

[33] Y. J. Kim, A. A. Awan, A. Muzio, A. F. C. Salinas, L. Lu, A. Hendy, S. Rajbhandari, Y. He, and H. H. Awadalla, "Scalable and efficient moe training for multitask multilingual models," 2021. [Online]. Available: https://arxiv.org/abs/2109.10465

[34] B. Klenk, N. Jiang, G. Thorson, and L. Dennison, "An In-Network Architecture for Accelerating Shared-Memory Multiprocessor Collectives," in *ACM/IEEE 47th Annual International Symposium on Computer Architecture*, ser. ISCA, IEEE. Washington, DC, USA: IEEE Computer Society, 2020, pp. 996–1009.

[35] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet Classification with Deep Convolutional Neural Networks," in *Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 1*, ser. NIPS'12. USA: Curran Associates Inc., 2012, pp. 1097–1105. [Online]. Available: http://dl.acm.org/citation.cfm?id=2999134.2999257

[36] Z. Lan, M. Chen, S. Goodman, K. Gimpel, P. Sharma, and R. Soricut, "ALBERT: A Lite BERT for Self-supervised Learning of Language Representations," in *Proceedings of the Seventh International Conference on Learning Representation*, ser. ICLR. OpenReview.net, 2019.

[37] D. Lepikhin, H. Lee, Y. Xu, D. Chen, O. Firat, Y. Huang, M. Krikun, N. Shazeer, and Z. Chen, "Gshard: Scaling giant models with conditional computation and automatic sharding," *arXiv preprint arXiv:2006.16668*, 2020.

[38] M. Lin, Q. Chen, and S. Yan, "Network in network," *CoRR*, vol. abs/1312.4400, 2013. [Online]. Available: http://arxiv.org/abs/1312.4400

[39] S.-C. Lin, Y. Zhang, C.-H. Hsu, M. Skach, M. E. Haque, L. Tang, and J. Mars, "The Architectural Implications of Autonomous Driving: Constraints and Acceleration," in *Proceedings of the Twenty-Third International Conference on Architectural Support for Programming Languages and Operating Systems*, ser. ASPLOS. New York, NY, USA: ACM, 2018, pp. 751–766. [Online]. Available: http://doi.acm.org/10.1145/3173162.3173191

[40] S. Liu, Q. Wang, J. Zhang, W. Wu, Q. Lin, Y. Liu, M. Xu, M. Canini, R. C. Cheung, and J. He, "In-network aggregation with transport transparency for distributed training," in *Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 3*, 2023, pp. 376–391.

[41] Y. Liu, M. Ott, N. Goyal, J. Du, M. Joshi, D. Chen, O. Levy, M. Lewis, L. Zettlemoyer, and V. Stoyanov, "RoBERTa: A Robustly Optimized BERT Pretraining Approach," 2019.

[42] P. Mattson, C. Cheng, C. Coleman, G. Diamos, P. Micikevicius, D. A. Patterson, H. Tang, G. Wei, P. Bailis, V. Bittorf, D. Brooks, D. Chen, D. Dutta, U. Gupta, K. M. Hazelwood, A. Hock, X. Huang, B. Jia, D. Kang, D. Kanter, N. Kumar, J. Liao, G. Ma, D. Narayanan, T. Oguntebi, G. Pekhimenko, L. Pentecost, V. J. Reddi, T. Robie, T. S. John, C. Wu, L. Xu, C. Young, and M. Zaharia, "MLPerf Training Benchmark," *CoRR*, vol. abs/1910.01500, 2019. [Online]. Available: http://arxiv.org/abs/1910.01500

[43] P. Micikevicius, D. Stosic, N. Burgess, M. Cornea, P. Dubey, R. Grisenthwaite, S. Ha, A. Heinecke, P. Judd, J. Kamalu, N. Mellempudi, S. Oberman, M. Shoeybi, M. Siu, and H. Wu, "Fp8 formats for deep learning," 2022. [Online]. Available: https://arxiv.org/abs/2209.05433

[44] Microsoft, "Turing-NLG: A 17-billion-parameter language model by Microsoft," *Microsoft Research Blog*, vol. 1, no. 8, 2020. [Online]. Available: https://www.microsoft.com/en-us/research/blog/turing-nlg-a-17-billion-parameter-language-model-by-microsoft/

[45] D. Moolchandani, J. Kundu, F. Ruelens, P. Vrancx, T. Evenblij, and M. Perumkunnil, "Amped: An analytical model for performance in distributed training of transformers," in *2023 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*. IEEE, 2023, pp. 306–315.

[46] NVIDIA, "NVIDIA TESLA V100 GPU ACCELERATOR," https://images.nvidia.com/content/technologies/volta/pdf/tesla-volta-v100-datasheet-letter-fnl-web.pdf, 2018.

[47] NVIDIA, "NVIDIA FasterTransformer," "https://github.com/NVIDIA/FasterTransformer/", 2020.

[48] NVIDIA, "NVIDIA A100 TENSOR CORE GPU," https://www.nvidia.com/content/dam/en-zz/Solutions/Data-Center/a100/pdf/nvidia-a100-datasheet-us-nvidia-1758950-r4-web.pdf, NVIDIA, 2021.

[49] ——, "NVIDIA Announces DGX GH200 AI Supercomputer," https://nvidianews.nvidia.com/news/nvidia-grace-hopper-superchips-designed-for-accelerated-generative-ai-enter-full-production, NVIDIA, 2023.

[50] J. Park, M. Naumov, P. Basu, S. Deng, A. Kalaiah, D. Khudia, J. Law, P. Malani, A. Malevich, S. Nadathur, J. Pino, M. Schatz, A. Sidorov, V. Sivakumar, A. Tulloch, X. Wang, Y. Wu, H. Yuen, U. Diril, D. Dzhulgakov, K. Hazelwood, B. Jia, Y. Jia, L. Qiao, V. Rao, N. Rotem, S. Yoo, and M. Smelyanskiy, "Deep learning inference in facebook data centers: Characterization, performance optimizations and hardware implications," 2018. [Online]. Available: https://arxiv.org/abs/1811.09886

[51] S. Pati, S. Aga, N. Jayasena, and M. D. Sinclair, “Demystifying BERT: System Design Implications,” in *2022 IEEE International Symposium on Workload Characterization (IISWC)*. IEEE, 2022, pp. 296–309.

[52] S. Pati, S. Aga, M. D. Sinclair, and N. Jayasena, “SeqPoint: Identifying Representative Iterations of Sequence-based Neural Networks,” in *IEEE International Symposium on Performance Analysis of Systems and Software*, ser. ISPASS. Washington, DC, USA: IEEE Computer Society, August 2020, pp. 69–80.

[53] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, and I. Sutskever, “Language Models are Unsupervised Multitask Learners,” *OpenAI Blog*, vol. 1, no. 8, 2019.

[54] C. Raffel, N. Shazeer, A. Roberts, K. Lee, S. Narang, M. Matena, Y. Zhou, W. Li, and P. J. Liu, “Exploring the limits of transfer learning with a unified text-to-text transformer,” 2019. [Online]. Available: https://arxiv.org/abs/1910.10683

[55] S. Rajbhandari, C. Li, Z. Yao, M. Zhang, R. Y. Aminabadi, A. A. Awan, J. Rasley, and Y. He, “Deepspeed-moe: Advancing mixture-of-experts inference and training to power next-generation ai scale,” 2022. [Online]. Available: https://arxiv.org/abs/2201.05596

[56] S. Rajbhandari, O. Ruwase, J. Rasley, S. Smith, and Y. He, “ZeRO-Infinity: Breaking the GPU Memory Wall for Extreme Scale Deep Learning,” in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, ser. SC ’21. New York, NY, USA: Association for Computing Machinery, 2021. [Online]. Available: https://doi.org/10.1145/3458817.3476205

[57] S. Rashidi, M. Denton, S. Sridharan, S. Srinivasan, A. Suresh, J. Nie, and T. Krishna, “Enabling compute-communication overlap in distributed deep learning training platforms,” in *2021 ACM/IEEE 48th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 2021, pp. 540–553.

[58] V. J. Reddi, C. Cheng, D. Kanter, P. Mattson, G. Schmuelling, C. Wu, B. Anderson, M. Breughe, M. Charlebois, W. Chou, R. Chukka, C. Coleman, S. Davis, P. Deng, G. Diamos, J. Duke, D. Fick, J. S. Gardner, I. Hubara, S. Idgunji, T. B. Jablin, J. Jiao, T. S. John, P. Kanwar, D. Lee, J. Liao, A. Lokhmotov, F. Massa, P. Meng, P. Micikevicius, C. Osborne, G. Pekhimenko, A. T. R. Rajan, D. Sequeira, A. Sirasao, F. Sun, H. Tang, M. Thomson, F. Wei, E. Wu, L. Xu, K. Yamada, B. Yu, G. Yuan, A. Zhong, P. Zhang, and Y. Zhou, “MLPerf Inference Benchmark,” in *ISCA*, 2020, pp. 446–459.

[59] S. Reed, K. Zolna, E. Parisotto, S. G. Colmenarejo, A. Novikov, G. Barth-Maron, M. Gimenez, Y. Sulsky, J. Kay, J. T. Springenberg, T. Eccles, J. Bruce, A. Razavi, A. Edwards, N. Heess, Y. Chen, R. Hadsell, O. Vinyals, M. Bordbar, and N. de Freitas, “A generalist agent,” 2022. [Online]. Available: https://arxiv.org/abs/2205.06175

[60] J. Ren, S. Rajbhandari, R. Y. Aminabadi, O. Ruwase, S. Yang, M. Zhang, D. Li, and Y. He, “Zero-offload: Democratizing billion-scale model training,” 2021. [Online]. Available: https://arxiv.org/abs/2101.06840

[61] J. Rettkowski and D. Göhringer, “Data stream processing in networks-on-chip,” in *2017 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, 2017, pp. 633–638.

[62] B. Rouhani, D. Lo, R. Zhao, M. Liu, J. Fowers, K. Ovtcharov, A. Vinogradsky, S. Massengill, L. Yang, R. Bittner, A. Forin, H. Zhu, T. Na, P. Patel, S. Che, L. C. Koppaka, X. Song, S. Som, K. Das, S. Tiwary, S. Reinhardt, S. Lanka, E. Chung, and D. Burger, “Pushing the limits of narrow precision inferencing at cloud scale with microsoft floating point,” in *Proceedings of the 34th International Conference on Neural Information Processing Systems*, ser. NIPS’20. Red Hook, NY, USA: Curran Associates Inc., 2020.

[63] Samsung, “GPU with HBM PIM,” https://semiconductor.samsung.com/newsroom/tech-blog/samsung-electronics-semiconductor-unveils-cutting-edge-memory-technology-to-accelerate-next-generation-ai/, 2022.

[65] M. Shoeybi, M. Patwary, R. Puri, P. LeGresley, J. Casper, and B. Catanzaro, “Megatron-LM: Training Multi-Billion Parameter Language Models Using Model Parallelism,” *CoRR*, vol. abs/1909.08053, 2019.

[64] K. Sangaiah, M. Lui, R. Kuttappa, B. Taskin, and M. Hempstead, “Snacknoc: Processing in the communication layer,” in *2020 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 2020, pp. 461–473.

[66] K. Simonyan and A. Zisserman, “Very deep convolutional networks for large-scale image recognition,” *CoRR*, vol. abs/1409.1556, 2014. [Online]. Available: http://arxiv.org/abs/1409.1556

[67] SK Hynix, “Gddr-pim,” https://news.skhynix.com/sk-hynix-develops-pim-next-generation-ai-accelerator/, 2022.

[68] S. Smith, M. Patwary, B. Norick, P. LeGresley, S. Rajbhandari, J. Casper, Z. Liu, S. Prabhumoye, G. Zerveas, V. Korthikanti *et al.*, “Using deepspeed and megatron to train megatron-turing nlg 530b, a large-scale generative language model,” *arXiv preprint arXiv:2201.11990*, 2022.

[69] M. Springer, P. Wauligmann, and H. Masuhara, “Modular Array-Based GPU Computing in a Dynamically-Typed Language,” in *Proceedings of the 4th ACM SIGPLAN International Workshop on Libraries, Languages, and Compilers for Array Programming*, 2017, p. 48–55.

[70] Y. Sun, S. Wang, Y. Li, S. Feng, H. Tian, H. Wu, and H. Wang, “ERNIE 2.0: A Continual Pre-training Framework for Language Understanding,” *CoRR*, vol. 1907.12412, 2019.

[71] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, “Going Deeper with Convolutions,” in *Computer Vision and Pattern Recognition (CVPR)*, 2015. [Online]. Available: http://arxiv.org/abs/1409.4842

[72] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, “Rethinking the inception architecture for computer vision,” *CoRR*, vol. abs/1512.00567, 2015. [Online]. Available: http://arxiv.org/abs/1512.00567

[73] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, “Attention Is All You Need,” in *Proceedings of the 31st International Conference on Neural Information Processing Systems*, ser. NeurIPS, 2017, p. 6000–6010.

[74] S. Verma, Q. Wu, B. Hanindhito, G. Jha, E. B. John, R. Radhakrishnan, and L. K. John, “Demystifying the mlperf training benchmark suite,” in *ISPASS*. IEEE, 2020, pp. 24–33.

[75] G. Wang, Y. Lin, and W. Yi, “Kernel Fusion: An Effective Method for Better Power Efficiency on Multithreaded GPU,” in *Proceedings of the 2010 IEEE/ACM Int’l Conference on Green Computing and Communications & Int’l Conference on Cyber, Physical and Social Computing*, 2010, p. 344–350.

[76] S. Wang, J. Wei, A. Sabne, A. Davis, B. Ilbeyi, B. Hechtman, D. Chen, K. S. Murthy, M. Maggioni, Q. Zhang *et al.*, “Overlap Communication with Dependent Computation via Decomposition in Large Deep Learning Models,” in *Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 1*, 2022, pp. 93–106.

[77] W. Xiong, , X. Huang, F. Seide, , and A. Stolcke, “Toward Human Parity in Conversational Speech Recognition,” *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, vol. 25, pp. 2410–2423, Sept 2017.

[78] Z. Yang, Z. Dai, Y. Yang, J. Carbonell, R. Salakhutdinov, and Q. V. Le, “XLNet: Generalized Autoregressive Pretraining for Language Understanding,” *CoRR*, vol. 1906.08237, 2020.

[79] A. H. Zadeh, Z. Poulos, and A. Moshovos, “Deep Learning Language Modeling Workloads: Where Time Goes on Graphics Processors,” in *IEEE International Symposium on Workload Characterization*, ser. IISWC, IEEE. Washington, DC, USA: IEEE Computer Society, 2019, pp. 131–142.

[80] B. Zheng, A. Tiwari, N. Vijaykumar, and G. Pekhimenko, “Ecornn: Efficient computing of lstm rnn training on gpus,” *arXiv preprint arXiv:1805.08899*, 2018.

[81] H. Zhu, M. Akrout, B. Zheng, A. Pelegris, A. Phanishayee, B. Schroeder, and G. Pekhimenko, “TBD: Benchmarking and Analyzing Deep Neural Network Training,” in *IEEE International Symposium on Workload Characterization*, ser. IISWC. Washington, DC, USA: IEEE Press, October 2018.