AMPeD: An **A**nalytical **M**odel for **Pe**rformance in **D**istributed Training of Transformers

Diksha Moolchandani, Joyjit Kundu, Frederik Ruelens, Peter Vrancx, Timon Evenblij, and Manu Perumkunnil
Interuniversity Microelectronics Centre (IMEC), Leuven, Belgium
{diksha.moolchandani, joyjit.kundu, frederik.ruelens, peter.vrancx, timon.evenblij, manu.perumkunnil}@imec.be

***Abstract*—Transformers are a class of machine learning models that have piqued high interest recently due to a multitude of reasons. They can process multiple modalities efficiently and have excellent scalability. Despite these obvious advantages, training these large models is very time-consuming. Hence, there have been efforts to speed up the training process using efficient distributed implementations. Many different types of parallelism have been identified that can be employed standalone or in combination. However, naively combining different parallelization schemes can incur significant communication overheads, thereby potentially defeating the purpose of distributed training. Thus, it becomes vital to predict the right mapping of different parallelisms to the underlying system architecture. In this work, we propose *AMPeD*, an analytical model for performance in distributed training of transformers. It exposes all the transformer model parameters, potential parallelism choices (along with their mapping onto the system), the accelerator as well as system architecture specifications as tunable knobs, thereby enabling hardware-software co-design. With the help of 3 case studies, we show that the combinations of parallelisms predicted to be efficient by *AMPeD* conform with the results from the state-of-the-art literature. Using *AMPeD*, we also show that future distributed systems consisting of optical communication substrates can train large models up to $4\times$ faster as compared to the current state-of-the-art systems without modifying the peak computational power of the accelerators. Finally, we validate *AMPeD* with in-house experiments on real systems and via published literature. The max. observed error is limited to 12%. The model is available here: https://github.com/CSA-infra/AMPeD**

I. Introduction

Deep learning algorithms are the basic building blocks of any AI application. An estimate of their growing importance can be gauged from their growing market value. The deep learning market was valued at 34.8 billion USD in 2021 and is predicted to be valued at 526.7 billion USD in 2030 growing at a CAGR of 34.3% [1]. In recent times, there have been particularly significant efforts to characterize [2] and accelerate the training of large language models such as transformers. The main reasons are the impressive scalability and accuracy achieved by transformer models on real-world tasks such as neural machine translation [3], sentiment analysis [4], automatic speech recognition [5], text classification [6], question answering, and visual object recognition [7].

However, this accuracy comes at a cost. The sizes of the transformer models and their datasets have grown exponentially over the years [8]. More specifically, the number of parameters in these transformer models has grown from 94 million in ELMo (2018) to 174 trillion in BaGuaLu (2021); nearly six orders of magnitude in 3 years. Training such large transformer models is computationally intensive and time-consuming. For example, when GPT-3 was first introduced in 2020, training it required 3.1 million GPU hours and would cost about $\$4.6$ million [9]. Moreover, the overall size of such large models is far beyond the physical memory capacity of a single accelerator (even for GPUs available today with large memories such as the 80GB Nvidia H100 cards [10]). Hence, there have been several attempts to accelerate the training process by distributing it to multiple accelerators.

The basic idea behind distributed training is to distribute the independent computations of the model onto multiple accelerators and enable parallel execution. Multiple parallelization strategies exist (see Section II), each one with advantages and disadvantages. Identifying the right type and degree of parallelism to be exploited under different constraints (such as budget, time, memory, and ease of implementation) can help in improving the training throughput considerably.

It is impractical to find the optimal type and degree of parallelism by performing actual training experiments given the above listed constraints. Additionally, distributed training of large ML models requires thousands of high-end GPUs; such systems are mostly not within the affordable budget of academia or research labs. Hence, most academic projects utilize cloud frameworks such as Microsoft Azure, Google Cloud Computing, or Amazon Web Services to train their proposed models. Nevertheless, executing these long-running experiments on cloud-hosted systems is also costly because users are billed per hour. In addition to the long training time and high cost of operation, the resulting energy usage and equivalent CO2 emissions are not in line with the goals of sustainable computing. Thus, it becomes extremely important to launch optimal parallel configurations that can train the model in an acceptable amount of time, budget, and energy. This makes it crucial to have a preliminary estimate of the training time for different configurations.

Thus, a *prediction of the training time for a given training algorithm and parallelism choices for a given system architecture with an accelerator design* becomes an essential element for the proposed distributed training workflow. The usage of performance predictors for scheduling jobs on cloud computing platforms to maximize a given metric (such as overall performance, throughput, or QoS) is well known [11], [12]. Performance prediction for deep learning workloads is also a well-established area. However, most of the research is focused on inference [13] and training of convolutional neural

networks [14], [15].

In this work, we propose an analytical performance model for distributed training of transformers. To the best of our knowledge, such a predictor is not a part of the training workflow (for transformers) that is usually executed on distributed systems. The main contributions of this work are as follows:

❶ We propose *AMPeD*, an analytical model for performance in distributed training of transformers that exposes many tunable knobs for design space exploration, such as machine learning model configurations, all the parallelism mappings (combinations of PP, DP, TP, and MoE), system architecture design choices, and the accelerator parameters.

❷ With the help of different case studies, we show that the predictions from *AMPeD* conform with the results from the state-of-the-art literature, in addition to providing insights and explanations into the training time breakdown. This further enables hardware-software co-design. Using *AMPeD*, we also show that future distributed system architectures consisting of optical communication substrates can train large models up to $4\times$ faster as compared to their training time on the current state-of-the-art systems.

❸ Finally, we validate *AMPeD* using in-house training experiments on smaller models and via published literature on large-scale distributed training approaches, demonstrating a maximal error of 12%.

II. Background

A. Transformers: Architecture and Training

Transformers are a class of deep neural networks that can learn context from sequential data such as words in a sentence. The architecture of a transformer primarily depends on an attention mechanism that finds correlations between different elements of a sequence to compute a representation of the sequence [16]. It can be either an encoder-decoder or encoder-only or decoder-only architecture, where each layer of these encoders and decoders consists of stacked self-attention and fully connected feed forward layers.

Training a transformer (or any artificial neural network) involves the following steps: ❶ loading (a batch of) input data, ❷ calculating the activations in the forward pass, ❸ calculating the loss at the end of the forward pass by comparing with the ground truth, ❹ calculating the gradients in the backward pass, and ❺ updating the weights using the gradients.

B. Types of Parallelisms in Distributed Training

In this section, we discuss the types of parallelisms that can be exploited to accelerate the training of transformers on distributed systems.

1) Data Parallelism (DP): This type of parallelism involves the distribution of input data set across multiple workers. Each worker sees only a part of the full data set, while weight matrices are replicated across ranks/workers leading to a higher memory footprint. There is no communication in the forward pass because it involves local matrix multiplications, however, gradient all-reduce is required as a collective operation during backpropagation.

An improvement over memory-intensive DP is Zero-DP [17]. The basic idea is to distribute the model parameters, gradients, and optimizer states across all workers and communicate them when necessary. This technique introduces extra communication in the forward and backward pass along with the already existing gradient all-reduce communication. However, it reduces the memory footprint by orders of magnitude, thereby supporting training with larger batch sizes that results in higher utilization of the accelerators.

2) Tensor Model Parallelism (TP): In Tensor Model Parallelism [8], the neural network weights are split while the training data is replicated across workers. Thus, every worker sees the same data but computes only a part of the activation or gradient, which is required to be communicated across accelerators in-between layers during forward and backward propagation. TP involves the splitting of the matrix multiplications within a transformer layer across multiple workers.

3) Pipeline Model Parallelism (PP): Pipeline model parallelism [8] involves splitting the layers of the neural network model across the workers. Activations from one set of layers, mapped to a worker, are passed on to the next set of layers, mapped to another worker. These sequential layers work on different data in parallel when the input batch is divided into microbatches that can be injected sequentially into the pipelined workers. Such a strategy can introduce *pipeline bubbles* or periods where an accelerator is idle, waiting for data to arrive from the previous accelerator in the pipeline. An overview of different techniques to reduce this idle time can be found in [8].

4) Mixture of Experts (MoE): MoE [18] refers to a collection of several experts (feed-forward neural networks), with the experts split across workers. A trainable gating network within a transformer block chooses a sparse combination of experts to process a given input. The basic idea here is to exploit conditional computation where only parts of the network are active, thereby leading to sparsity. Thus, the number of model parameters to express the training data explodes by several orders of magnitude with a minimal increase in the computational cost.

III. Related Work

There have been multiple efforts in recent years toward predicting the performance of deep learning training workloads. Yan *et al*. [19] proposed a performance model for the training of deep neural networks on distributed systems, however, the formulation takes into account only convolutional and fully connected layers. On similar lines, Qi *et al*. [20] proposed an analytical performance prediction model for training CNNs on distributed infrastructure that relies on the deterministic computations associated with CNNs and mapping them onto the underlying system and communication strategies. Similarly, Gianniti *et al*. [14] proposed a linear regression-based performance prediction model for training individual CNN layers on a single accelerator. Another recent work by Yu *et al*. [21] proposed a machine learning-based performance model, aided by a heuristic based on the ratios of memory

bandwidth and compute units of the two accelerators, to predict the training times of DNNs on another accelerator while using the characteristic behavior of the machine learning model on a given accelerator. Rashidi et al. [22] proposed a simulator for hardware software co-design exploration of deep learning training algorithms, however they primarily focus on the effect of different network topologies and communication type on the training time.

In contrast, *AMPeD* is significantly different from all the previous works. It analytically models the performance for distributed training of transformers on a given system architecture, accelerator design, and mapping of different parallelisms to the underlying system, and the model parameters. Such modeling enables hardware-software co-design of the transformers models and the system architecture. Though *AMPeD* also utilizes the inherent determinism associated with training these language models, mapping different types of parallelism for time-efficient training is a challenging task.

IV. The Making of *AMPeD*

AMPeD assumes a distributed system with multiple nodes, where each node is further composed of several homogeneous accelerators. The accelerators within a node communicate with each other using intra-node links and those across different nodes communicate using inter-node links. *AMPeD* assumes that the training time is dominated by the computation time and communication time of these accelerators (and not of their host processors). It incorporates the effect of parallelism on computation time and also the communication overheads incurred as a result of this. In the following sections, we consider all four kinds of parallelism – DP, TP, PP and MoE and their impact on training times.

The overall training time is calculated in Eq. 1, where N_{batch} represents the number of batches in the training data, $\sum_l$ represents the sum over all layers of the neural network, $U(l)$ and $M(l)$ represent computation time per layer and communication time per layer l, respectively. The subscript letter indicates the forward pass (f), the backward pass (b), the weight update phase (w), or the all-reduce of gradients between the forward and backward pass (g). $W(l)$ stands for the waiting time per layer caused by the bubbles of pipelining (if PP is considered) as explained in later sections. N_{TP}, N_{DP}, and N_{PP} represent the degrees of tensor model parallelism, data parallelism, and pipeline parallelism, respectively. In Eq. 1, we express the total training time as the sum of the computation time (scaled down by the number of workers involved with different parallelism choices), and communication time, summed over all the layers and scaled by the number of batches.

$$\text{Time} = N_{\text{batch}} \sum_l \Big[\frac{U_f(l) + U_b(l) + U_w(l)}{N_{\text{TP}}\ N_{\text{DP}}\ N_{\text{PP}}} + M_f(l) + M_b(l) + M_g(l) + W(l) \Big] \tag{1}$$

A. Estimating Forward Pass Computation Time

The computation time for the forward pass $U_f(l)$ of a transformer layer l is calculated by summing over the contributions of all the sublayers within it (denoted by i; e.g., the Multi-Layer Perceptron or the attention sublayer), – we multiply the number of operations (either MACs ($N_{\text{MAC}}(l,i)$), or non-linear operations($N_{\text{nonlin}}(l,i)$), with the reciprocal of the throughput of the accelerator for that operation, C_{MAC} and C_{nonlin}, respectively. To quantify the time for which the computation unit is busy, the throughput is scaled with the maximum precision of the operands divided by the hardware-determined precision of the functional unit. Here, S_p, S_{act}, S_{nonlin} denote parameter precision, activation precision, and nonlinear operation precision. $S_{\text{FU}_{\text{MAC}}}$ and $S_{\text{FU}_{\text{nonlin}}}$ denote the hardware-determined precision of the functional unit; *ceil* represents the ceiling function.

The reciprocal of the throughput is calculated based on the accelerator design parameters, and a microbatch efficiency factor $\text{eff}(ub)$. The accelerator design parameters include the frequency f, the number of cores N_{cores}, the number of functional units (N_{FU} and $N_{\text{FU}_{\text{nonlin}}}$) and their respective widths (W_{FU} and $W_{\text{FU}_{\text{nonlin}}}$), expressed in the precision of the functional units as described before. We scale the peak performance of the MAC units by $\text{eff}(ub)$ to capture the utilization of compute cores – this can be obtained by fitting the experimental data based on the application and the underlying hardware. Empirically, a functional form of $\frac{a.ub}{b+ub}$, allows a good fit until a critical microbatch size, with a and b being functions of the application and the underlying system architecture under consideration [23]. Note that when $ub \gg 1$, performance can decay due to generalization gaps [24].

$$U_f(l) = \sum_i N_{\text{MAC}}(l,i)\ C_{\text{MAC}}\ \text{ceil}\left[\frac{\max(S_p(l,i), S_{\text{act}}(l,i))}{S_{\text{FU}_{\text{MAC}}}}\right] + N_{\text{nonlin}}(l,i)\ C_{\text{nonlin}}\ \text{ceil}\left[\frac{S_{\text{nonlin}}(l,i))}{S_{\text{FU}_{\text{nonlin}}}}\right], \tag{2}$$

$$C_{\text{MAC}} = (f\ N_{\text{cores}}\ N_{\text{FU}}\ W_{\text{FU}}\ \text{eff}(ub))^{-1} \tag{3}$$

$$C_{\text{nonlin}} = (f\ N_{\text{FU}_{\text{nonlin}}}\ W_{\text{FU}_{\text{nonlin}}})^{-1} \tag{4}$$

B. Estimating Forward Pass Communication Time

Distributing calculations over multiple accelerators incurs communication. We incorporate all the methods of distributing training of large transformers (as explained in Section II) in Eq. 5 for the forward pass. We consider the intra- and inter-node parallelisms separately since they use different communication bandwidths. The communication incurred as a result of tensor parallelism within a node and across nodes is added assuming a hierarchical all-reduce operation, where the activations are first reduced within the node and then across nodes depending on the mapping. For pipeline parallelism, communication is always from one accelerator to the next. The pipeline is only as fast as its slowest step, hence we take the maximum of the intra-node and inter-node communication overhead. We include the communication overhead as a result of the mixture of experts parallelism, $M_{f_{\text{MoE}}}$. Note that regular DP does not have any communication in the forward pass. For Zero-powered data parallelism, we add an overall overhead factor, $M_{f_{DP}}$ [17].

$$M_f(l) = (1 + M_{f_{\mathrm{DP}}})[M_{f,\mathrm{TP,intra}}(l) + M_{f,\mathrm{TP,inter}}(l) + M_{f,\mathrm{MoE}}(l) + \max(M_{f,\mathrm{PP,intra}}(l), M_{f,\mathrm{PP,inter}}(l))] \quad (5)$$

1) Tensor Parallel Communication Time: Eq. 6 shows the calculation of the forward pass communication time as a result of intra-node tensor parallelism. Here, C_{intra} represents the latency of the intra-node communication link, and BW_{intra} represents the bandwidth of the intra-node link. T_{intra} is an intra-node topology factor describing the number of communication steps required for a certain topology divided by the number of accelerators communicating [25]. For example, an all-reduce of the tensor parallel accelerators implemented on a ring topology within a node results in a topology factor of 2 $(N_{\mathrm{TP,intra}} - 1)/N_{\mathrm{TP,intra}}$. The number of activations that needs to be communicated for intra-node tensor parallelism is represented by $N_{\mathrm{act,TP}}(l)$. A transformer layer using tensor parallelism incurs two all-reduce steps, one for the attention sublayer, and the other for the MLP sublayer. Both steps communicate the same amount of data, which can be calculated with the effective batch size (b), sequence length (s), and hidden layer size (h) [8].

Inter-node tensor parallel communication is calculated similarly, using inter-node latency, bandwidth, and topology factor. The number of activations that need to be communicated is 0 when no inter-node TP is used, otherwise, it is $N_{\mathrm{act,TP}}(l)$.

$$M_{f,\mathrm{TP,intra}}(l) = C_{\mathrm{intra}}\ T_{\mathrm{intra}}\ N_{\mathrm{TP,intra}} + N_{\mathrm{act,TP}}(l)\ \frac{S_{\mathrm{act}}}{BW_{\mathrm{intra}}}\ T_{\mathrm{intra}}, \quad (6)$$

where $N_{\mathrm{act,TP}}(l) = 2bsh$.

2) Pipeline Parallel Communication Time: The communication time incurred as a result of pipeline model parallelism is estimated similarly. If intra-node PP is used, we estimate the communication overhead per layer as shown in Eq. 7, where every accelerator communicates $N_{\mathrm{act,PP}}(l)$ activations to the next accelerator in sequence. The pipeline topology is fixed (one-to-one links), so no topology factor is needed. Since pipeline communication happens in parallel for all layers in the model, its overhead is independent of the number of layers. We include a factor $\frac{1}{L}$ to scale the overhead per layer for use in Eq. 1. L is the total number of layers in the transformer model. Inter-node PP overhead is estimated in exactly the same way, using different values for the latency and bandwidth variables.

$$M_{f,\mathrm{PP,intra}}(l) = \frac{1}{L}\left[C_{\mathrm{intra}} + N_{\mathrm{act,PP}}(l)\ \frac{S_{\mathrm{act}}}{BW_{\mathrm{intra}}}\right], \quad (7)$$

where $N_{\mathrm{act,PP}}(l) = bsh$.

C. Estimating the Impact of Pipeline Bubbles

The waiting time due to pipeline bubbles can be summarized in Eq. 8. R describes the ratio of non-overlapping bubbles in the deployed pipeline parallel scheme versus the number of non-overlapping bubbles in naive pipeline parallelism [26], allowing to easily estimate more efficient pipeline strategies [8]. N_{ub} is the number of microbatches per (mini)batch. The equation represents the waiting time $W(l)$ for $N_{\mathrm{PP}} - 1$ idle pipeline steps, when there are N_{ub} useful forward and backward pipeline steps. Each pipeline step works on a microbatch, with a forward pass and backward pass. So, the duration of each step ($\sim$ Eq. 1) is determined by forward and backward pass computation time, scaled by the total amount of parallel workers, and the forward and backward pass communication time. Note that weight updates and gradient synchronization happen outside of the pipeline.

$$W(l) = R\ \frac{N_{\mathrm{PP}} - 1}{N_{ub}} \times \left[\frac{U_f(l) + U_b(l)}{L\ N_{\mathrm{TP}}\ N_{\mathrm{DP}}\ N_{\mathrm{PP}}} + M_b(l) + M_f(l)\right] \quad (8)$$

D. Mixture of Experts Communication Time

Using MoE induces two all-to-all communication patterns per layer that include experts [18]. If no experts are used in a layer, there is no extra communication time. We estimate this overhead in Eq. 9. N_{nodes} represents the number of multi-accelerator nodes in the system. The topology factor T_{MoE} now represents the communication steps required to perform an all-to-all operation on a certain topology. In a default pairwise exchange case, it is equal to $(N_{\mathrm{nodes}} - 1)/N_{\mathrm{nodes}}$. A part of the communication will go to accelerators in the same node, and another part of it will go to accelerators in a different node. We assume a uniform random distribution and perfect load-balancing between experts. Thus, the accelerator communicates with another accelerator in the same node with a probability of $1/N_{\mathrm{nodes}}$, and with an accelerator in a different node with a probability of $(N_{\mathrm{nodes}}-1)/N_{\mathrm{nodes}}$. $N_{\mathrm{act,MoE}}(l)$ is the same as $N_{\mathrm{act,PP}}(l)$ for the layers where experts are used.

$$M_{\mathrm{f,MoE}}(l) = 2\ C_{\mathrm{inter}}\ T_{\mathrm{MoE}}\ N_{\mathrm{nodes}} + 2\ N_{\mathrm{act,MoE}}(l)\ S_{\mathrm{act}}\ T_{\mathrm{MoE}} \times \left[\frac{1}{N_{\mathrm{nodes}}\ BW_{\mathrm{intra}}} + \frac{N_{\mathrm{nodes}} - 1}{N_{\mathrm{nodes}}\ BW_{\mathrm{inter}}}\right] \quad (9)$$

E. Estimating Backward Pass Communication Time

Estimating communication time for the backward pass is very similar to the forward pass, albeit activations are replaced by error and gradient calculations. For brevity, we choose to omit the corresponding equations.

F. Estimating Gradient Communication Time

After gradients are calculated on each accelerator, they need to be reduced over all accelerators, so that each accelerator can update the weights it is responsible for. We assume a hierarchical all-reduce implementation in Eq. 10, that is, summing the time required to first reduce inside a node with the time required to reduce between nodes. Eq. 11 shows the time taken for intra-node gradient reduction, $\sim$ Eq. 6, also estimating an all-reduce communication type. The main difference is that we are using data parallel accelerators in this case and communicating gradients instead of activations. Hence, we use the number of communicating accelerators $N_{\mathrm{DP,intra}}$, the number of gradients to communicate N_g, and the size of each gradient S_g. Similar calculations are needed for inter-node gradient communication as well.

$$M_{\mathrm{g}}(l) = M_{\mathrm{g,intra}} + M_{\mathrm{g,inter}} \quad (10)$$

$$M_{\mathrm{g,intra}}(l) = C_{\mathrm{intra}}\ T_{\mathrm{intra}}\ N_{\mathrm{DP,intra}} + N_{\mathrm{g}}(l)\ \frac{S_{\mathrm{g}}}{BW_{\mathrm{intra}}}\ T_{\mathrm{intra}} \quad (11)$$

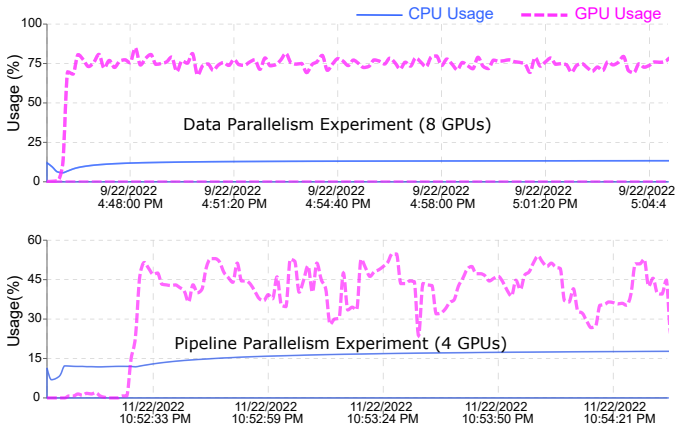

Fig. 1. Example runs of validation experiments for DP and PP on 8 and 4 GPU node configurations, respectively.

G. Estimating the Weight Update Time

Eq.12 shows the computation time required to update the weights after gradient calculation and reduction. $N_{MAC}(l)$ represents the number of weights to be updated in layer l. Since weight updates are also MAC operations, we multiply the number of weights in the model with the reciprocal of the MAC throughput C_{MAC}.

$$U_w(l) = C_{\text{MAC}}\ N_{\text{MAC}}(l) \tag{12}$$

To conclude, *AMPeD* is generic enough to be ported to other DNN models. The primary differences involved here will be in the calculation of N_{MAC}, N_{act}, N_{err}, N_g, and N_{nonlin}. Also in our modeling, MoE is specific to transformers. However, *AMPeD* is parameterizable enough to turn off this feature.

V. Validation

TABLE I
Experimental setup to validate DP and PP

Node	HGX-2 (Upto 16 accelerators)
Accelerator	Nvidia V100 (GV100) SXM3
Architecture	Volta (5120 CUDA Cores, 640 Tensor Cores)
Clock (Base/Boost)	1290 MHz/1530 MHz
Process Tech.	12nm (TSMC)
Memory Specifications	
Memory Type	HBM2
Memory Size (Specified/Available)	32 GB/31.75 GB
Memory Bus	4096
Memory Clock	876 MHz (1752 MBps effective)
Memory Bandwidth	897 GB/s
Accelerator TDP	250W
Host Memory	1.6 TB
Intra-node Network	NVLink Interconnect+NVSwitch

A. Validating the Impact of DP

In our first validation experiment with DP, we train *minGPT* (85M parameters) [27] on a single HGX-2 node. Table I details the system specifications for our experimental setup. Fig. 1 shows the CPU and GPU usage observed while validating DP and PP on 8 and 4 GPUs (within a single node), respectively. *minGPT* is a transformer language model with 12 hidden layers and 12 attention heads for each attention layer in the transformer encoder. The dimensionality of the embedding and hidden states is set to 768.

We monitor the training times for a fixed number of batches and capture the normalized training times for 1, 2, 4, 8, and 16 GPUs (within a single node) as shown in Fig. 2a). We adjust the batch size if needed to fit into the GPU memory for optimal batch efficiency. For the predictions, we use the average microbatch efficiency as obtained during the runtime of the experiment. We observe that the trends obtained from the hardware experiments (labeled as *Experimental* in the figure) match well with those obtained from *AMPeD* (labeled as *Predicted* in the figure).

B. Validating the Impact of PP

For the second validation experiment of *AMPeD*, we train minGPT with PP [28]. In these experiments, we use a variant of the minGPT model with 16 hidden layers (increased the layers to utilize PP to 16 GPUs) and 8 attention heads for each attention layer in the transformer encoder (1.24B parameters). The dimensionality of the embedding and hidden states is set to 1024 (set to achieve an optimal training time). For our experiments, we train the model using the corpus of Wikipedia articles [29].

Similar to the previous experiment, Fig. 2b) compares relative training times on 2, 4, 8, and 16 GPUs with respect to the training time on 2 GPUs. For these experiments, we set the number of microbatches to be equal to the pipeline degree or the number of GPUs used. We see that *AMPeD* captures the experimental trend accurately. The performance saturation from 8 to 16 GPUs is primarily due to the specifics of the implementation we use for the experiments– it is bottlenecked by the memory of the last GPU (all the microbatches are gathered at the last GPU) and thus, does not allow us to scale the global batch size with increasing the number of GPUs. We further validate the performance of PP at scale using published data in the following section.

C. Validating against Published Results

In this section, we validate *AMPeD* at scale against the published data in the literature. Since the training data from the published literature often lacks exact parameter details, we limited our validation to the cases with sufficient detail [8], [26]. First, we consider different language models (GPT) with a varied number of parameters. In Table II, we compare the performance (TFLOP/sec/GPU) of published data and the predictions from *AMPeD*. The data correspond to language models with 145B, 310B, 530B and 1T [8]. We observe that *AMPeD* is within 12% of the published data.

TABLE II
Comparison of performance: *AMPeD* vs published data [8]

Model Size	TP	PP	DP	AMPeD TFLOPs/GPU	Published TFLOPs/GPU	Error (%)
145B	8	8	24	147	148	0.6
310B	8	16	12	162	155	4.5
530B	8	35	9	148.6	163	8.8
1T	8	64	6	144.3	163	11.47

Note that we cannot conclude that the errors increase with the model size. As can be observed that the degree of PP also increases with the model size. However, the published results [8] have used interleaved PP that allows the overlapping of pipeline bubbles. In our model, we capture the effect of overlapping bubbles using the factor R as shown in Eq. 8.

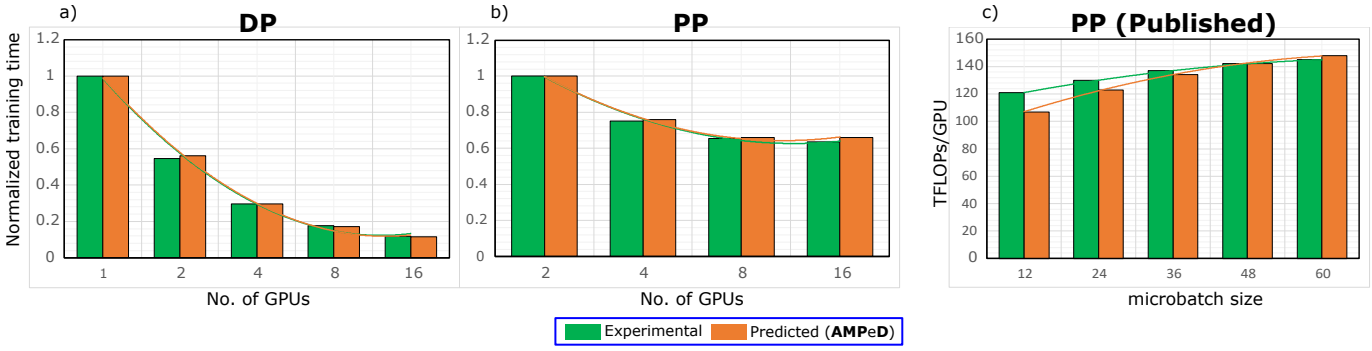

Fig. 2. Comparison of the normalized training time versus number of GPUs used between experiments and *AMPeD* for minGPT in a a) data parallel setting, b) pipeline parallel setting. c) Performance measured in TFLOP/S/GPU as a function of the batch size for a GPT model with 175 billion parameters on 96 GPUs with pipeline parallelism: published results [8] and *AMPeD*

For the estimations in Table II, we set $R = 1$, and thus do not consider overlapping bubbles. Hence, with increasing PP our bubble time also increased and there is a larger visible error. Nevertheless, R can be tuned to fit the data or can be modeled in more detail as a function of pipeline stages and interleaving.

Next, we verify the same performance metric as a function of batch size for GPT-3 model with 175B parameters on 96 GPUs as presented in [8], using only pipeline parallelism. We observe that *AMPeD* captures the performance saturation when increasing the microbatch size. The error is $\approx 11\%$ for a microbatch size of 12 and converges to merely $\approx 2\%$ for a microbatch size of 60 as shown in Fig.2c).

TABLE III
COMPARISON OF THE NORMALIZED TRAINING THROUGHPUT FOR GPIPE IMPLEMENTATION (PP) ON P100 GPUS WITH PCI-E USING 32 MINIBATCHES (M).

number of GPUS	2	4	8
M=32 (published [26])	1	1.8	3.3
M=32 (prediction)	1	1.84	3.19

We further validate the GPipe implementation of a 24-layer transformer model on NVIDIA P100 GPUs connected via PCIe3.0. Table. III compares the computed speedup with 32 microbatches for multiple GPUs with the normalized ones stated in [26]. As presented in [26], we tune the microbatch size according to the available memory of P100. With the resulting efficiency, we are able to match the published results within 12% error bar.

To conclude, our work has been validated taking into account the most relevant parallelism combinations that are being used in the current deployments [8], [26] and will be used in future as well. The only parallelism that we did not validate with the published results is MoE. The primary reason was the unavailability of exact details of the model/system parameters or the difference in the algorithmic implementation. We also specifically tried to extract these details from [30], [31]. However, these implementations were different than those considered in *AMPeD*.

We have shown that *AMPed* can use empirically derived efficiency factors to accurately predict the training time. A predictive model for $\text{eff}(ub)$ is left for future work.

VI. CASE STUDY I: OPTIMIZING PARALLELISM CONFIGURATION FOR A GIVEN SYSTEM

TABLE IV
ACCELERATOR CONFIGURATIONS USED IN THE EXPLORATION

Hardware	Freq. (f) cycles/s	N_{cores}	N_{FU}	W_{FU}	$N_{FU_{nonlin}}$	$W_{FU_{nonlin}}$	BW_{intra} bits/s
Nvidia A100	1.41E+09	108	4	512	192	4	2.4E+12
Nvidia H100	1.8E+09	132	4	1024	320	4	3.6E+12

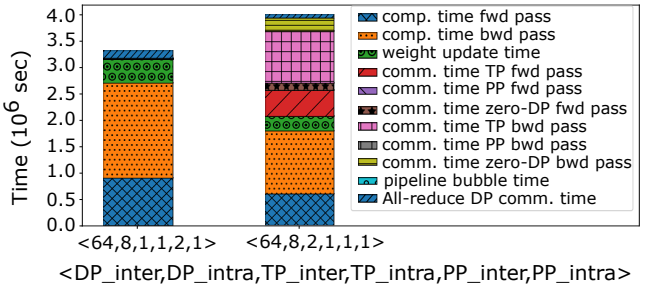

Fig. 3. Training time breakdown for two example configurations: one with PP and other with TP in inter-node accelerators

In this section, we perform a design space exploration using *AMPeD*. We consider a system with a total of 1024 accelerators distributed across 128 nodes, where each node consists of 8 accelerators. The accelerator is modeled after an Nvidia A100 GPU [32], connected via NVLink. Table IV shows the details used in the calculations. The nodes are connected over an HDR Infiniband network. We perform all the experiments for the Megatron model with 145 billion parameters taken from [8]. For this exhaustive exploration, we consider all possible combinations of data, pipeline, and tensor parallelism in intra-node and inter-node accelerators.

We use *AMPeD* to explore the training times for batch sizes equal to 4096, 8192, and 16384. We require large batch sizes in distributed training as the degrees of DP and PP negatively affect the eventual microbatch size. A large enough microbatch results in higher microbatch efficiency per accelerator. However, convergence and accuracy issues might arise as large batch sizes tend to converge to sharp minima that lead to poorer generalization [33]. Nevertheless, there have been works to mitigate this issue by keeping the variance of

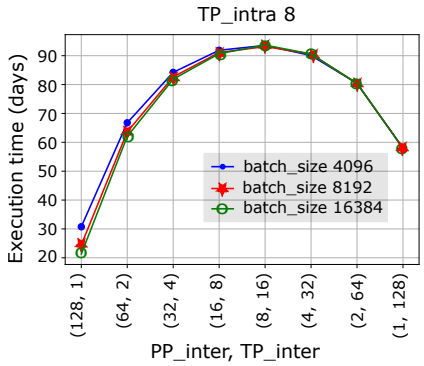

Fig. 4. TP intra-node, (PP, TP) inter-node

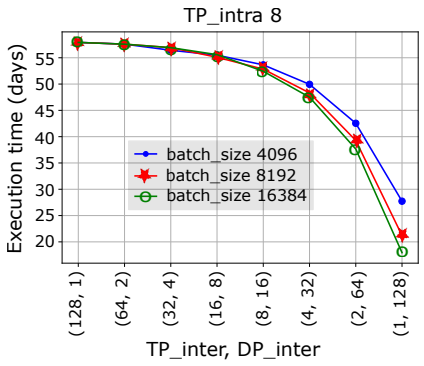

Fig. 5. TP intra-node, (TP, DP) inter-node

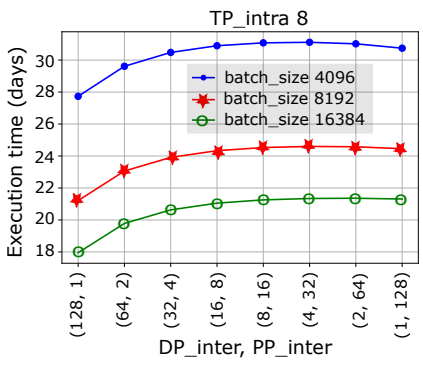

Fig. 6. TP intra-node, (PP, DP) inter-node

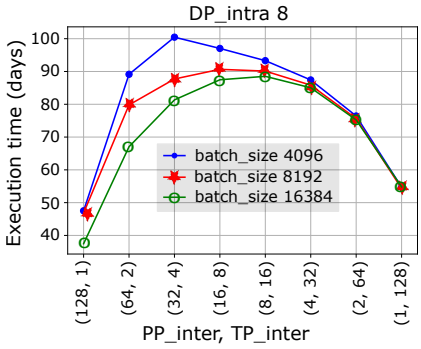

Fig. 7. DP intra-node, (PP, TP) inter-node

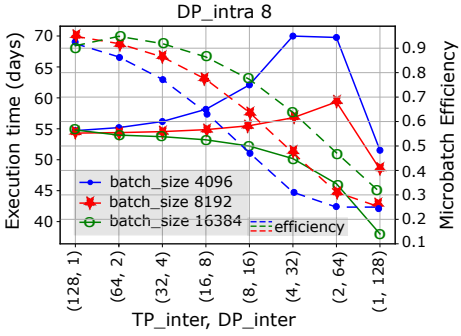

Fig. 8. DP intra-node, (TP, DP) inter-node

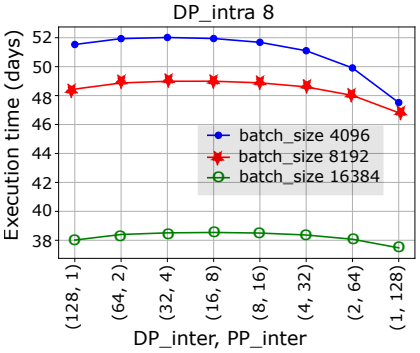

Fig. 9. DP intra-node, (PP, DP) inter-node

larger batches constant while matching the accuracy of small batch sizes [34], [35]. In our exploration, we consider batch sizes up to 16384, assuming minimal impact on convergence or accuracy.

A. *AMPeD Training Time Breakdown*

AMPeD has the capability to show a detailed breakdown of the time spent in computation and communication due to TP, PP, and DP individually. Fig. 3 shows a breakdown of training time for two example configurations, where $\mathrm{DP}_{\mathrm{inter}} = 64$, and $\mathrm{DP}_{\mathrm{intra}} = 8$. For the first config, $\mathrm{PP}_{\mathrm{inter}} = 2$ and for the second config, $\mathrm{TP}_{\mathrm{inter}} = 2$. From the breakdown example, we can observe that the pipeline bubble time in the first configuration is negligible compared to the communication overheads in the second configuration. We use insights from this breakdown to understand the design space exploration in the following sections.

B. *Exploration with* PP *in Intra-node Accelerators*

For PP in intra-node and TP in inter-node accelerators, the training time is quite large, ~ 90 days. Upon replacing TP with a combination of PP and DP in inter-node accelerators, the training time drops by around 50% for all cases. The primary reason is the huge communication time incurred in executing TP in inter-node accelerators as a result of two all-reduce communications of activations at each transformer layer over the slow inter-node network. Even though the training time improves on distributing DP/PP in inter-node accelerators, it is still large, as the microbatch efficiency is quite low (approx. 31%) even for a larger batch size of 16384; it is even lower for smaller batch sizes. The microbatch size is the batch size shrinked by a factor of $\mathrm{PP}_{\mathrm{intra}}$ in the former case (TP in inter-node) and $\mathrm{PP}_{\mathrm{intra}} \times \mathrm{PP}_{\mathrm{inter}} \times \mathrm{DP}_{\mathrm{inter}}$ in the latter case (combination of DP and PP in inter-node).

C. *Exploration with* TP *in Intra-node Accelerators*

Fig. 4, 5, and 6 show the training time with TP in intra-node accelerators. We observe that with pure PP or DP in inter-node accelerators, the obtained training time is small ($\sim 18 - 21$ days), while it is quite long (~ 57 days) with TP in inter-node accelerators. The impact of bubble time as a result of having PP in inter-node accelerators is orders of magnitude smaller than the impact of communication time as a result of TP. Hence, Fig. 4 shows almost $3\times$ increase in the training time in scaling down PP and scaling up TP by 2 (both done simultaneously to keep the number of accelerators the same). The microbatch efficiency is high for all cases because using TP in intra-node accelerators limits the degree of DP or PP to 128 for our setup, which is sufficient to ensure a high microbatch efficiency.

Another useful insight is that PP performs slightly worse than DP in inter-node accelerators. For example, for 16384 batch size, the training time using PP for inter-node accelerators is ~ 21 days, while the training time using DP for inter-node accelerators is ~ 18 days. Even though DP incurs an all-reduce overhead while PP incurs pipeline bubbles, the time spent in DP all-reduce is 2 orders of magnitude smaller than the pipeline bubble time.

D. *Exploration with* DP *in Intra-node Accelerators*

In Fig. 7, 8, and 9, we show the training time when DP is implemented in the intra-node accelerators. We observe that

the curves for training times in Fig. 7 start to merge for TP $>$ PP in the inter-node accelerators. The primary reason is the dominance of the communication time over computation time with increasing TP, which is not impacted by the batch size.

We also observe in Figure 8 that for TP+DP in inter-node accelerators, the training time curves show different behavior for different batch sizes even when the microbatch efficiency behaves similarly. First, we observe the figure till the configuration $(TP_{inter}, DP_{inter}) = (4, 32)$. For smaller batch sizes of 4096 and 8192, the training time increases as inter-node DP increases, albeit the growth is slow for the batch size 8192. Interestingly, for a batch size of 16384, the training time decreases with an increase in the inter-node DP. For all batch sizes, communication time (not shown) reduces similarly with decreasing inter-node TP. However, computation time goes up due to decreasing microbatch efficiency, which decays at a faster pace for smaller batch sizes. Note that because the microbatch efficiency curve has a fixed lower limit of 25% in our case, the trend for training time changes suddenly after the configuration $(TP_{inter}, DP_{inter}) = (4, 32)$ – this is an artifact of the efficiency function we choose.

Comparing Fig. 6 and 9, we observe that the training time is quite high $(36 - 38$ days) with DP in intra-node accelerators, while it is nearly $18 - 21$ days with TP in intra-node accelerators for a batch size of 16384. The primary reason for this behavior is a smaller microbatch size that occurs as a result of a high degree of DP leading to low microbatch efficiency (explained in Section VI-B). Hence, the microbatch efficiency is only 30% for the former case (DP in intra-node) and up to 80% for the latter case (TP in intra-node).

E. Case Study I *Conclusions*

Based on our explorations, we can conclude the following: ❶ Large batch sizes are required (due to microbatch efficiency) to keep performance high when parallelizing on large distributed systems using PP or DP. ❷ TP effectively parallelizes without lowering microbatch efficiency but is communication-intensive. Hence, it is very efficient to utilize for accelerators connected with high intra-node bandwidth, but very inefficient over slower inter-node communication links. ❸ DP and PP are better options ($2\times$ faster training than TP) to exploit in inter-node accelerators connected via slower inter-node communication links. ❹ Comparing the same degree of pure DP and pure PP in inter-node accelerators, the time spent in DP all-reduce is $2\times$ less than the pipeline bubble time if the model and system configuration are along the lines of our experimental setup. ❺ For the same inter-node configurations, TP in intra-node accelerators is $2\times$ faster than PP, DP, or a combination of DP and PP.

VII. Case Study II: Exploring Inter-node Parallelism for Low-end Systems

From the previous case study, one might conclude that employing DP for inter-node parallelism is always beneficial over using PP (see conclusion ❹ in Section VI-E). However, this is not always the case as the best parallelism strategy depends on the many parameters included in *AMPeD*. For example, we can explore low-end system architectures, with the same total amount of accelerators but fewer accelerators and network cards per node and using EDR network cards instead of HDR network cards. Such low-end system architectures are often more commonly available from cloud service providers as compared to high-end systems with many intra-node accelerators and very high-speed network cards. Fig. 10 shows the training time for the Megatron 145B model using a batch size of 8192 and TP for intra-node parallelism. We compare the training times using DP versus PP for inter-node accelerators when using 1, 2, 4, and 8 accelerator(s) + network card(s) per node. When using 1 accelerator and 1 network card in every node, the all-reduce communication pattern in DP takes a significant amount of time, whereas PP only involves point-to-point communication of the activations or gradients from one layer to the next. In this case, using PP for inter-node parallelism results in 80% higher performance. When upgrading to 2 accelerators/network cards per node, the difference is greatly reduced to a 17% performance benefit for PP. For the other configurations, DP is the better choice. Interestingly, for the 4 accelerators/network cards per node configuration, while the PP configuration takes around 1 day ($\sim 4\%$) longer to train the model, it is most likely a more energy-efficient configuration. Using PP introduces pipeline bubbles ($\sim 11\%$ in this case), in which accelerators are idling. During this idle time, the required power is greatly reduced. If the power savings of the system during these bubbles is larger than the extra energy cost due to the increased training time, this is still a more energy-efficient configuration. In this case, the lower power state should use less than $\sim 30\%$ of the power of the system during full execution, which is a realistic scenario. We leave power modeling and more detailed analysis for future work.

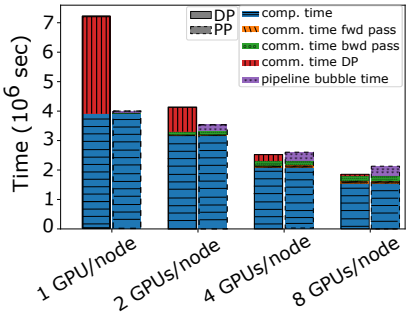

Fig. 10. Execution/training time for Megatron 145B model with DP/PP in inter-node, TP in intra-node, and different number of accelerators/EDR network cards per node keeping the total accelerators constant to 1024

From this example exploration, we conclude that for low-end system architecture configurations, the optimal parallel strategy choice can be different than for high-end systems.

VIII. Case Study III: Exploring Future Distributed Systems using Optical Communication Substrates

AMPeD can also be used to explore the design space for fu-

ture systems. This section gives such an example, specifically on how distributed training time can be impacted by emerging interconnect technologies such as optical communication substrates [36] enabled by silicon photonics [37], [38]. Using *AMPeD*, we explore the performance of distributed training systems where the accelerators in a node are connected by an optical communication substrate. We assume the accelerators do not include optical communication on the die, so the optical substrate is responsible for converting electrical to optical signals. Under these assumptions for the experimental setup, three potential optimizations can be exploited to increase performance [36]: ❶ *Opt.1* When connecting multiple nodes, the inter-node bandwidth per accelerator is increased because now multiple substrates are easily connected with optical fibers, bypassing the network card interface. We assume one fiber attached per accelerator on the edge of the substrate ❷ *Opt.2* The substrate makes it easier to connect more accelerators in the same node at their full off-chip bandwidth. ❸ *Opt.3* Off-chip bandwidth can be increased for future accelerator designs, as the electrical signals from the accelerator have to travel a very short distance to the substrate which converts the signals to the optical domain.

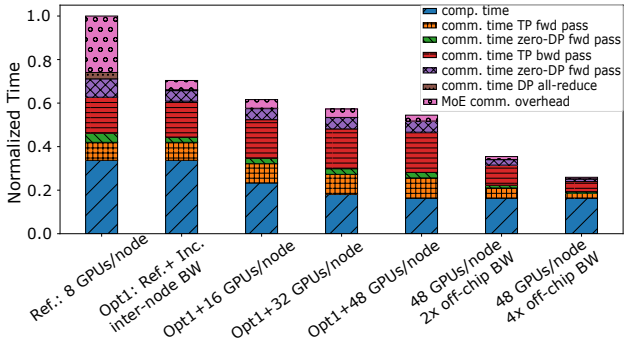

Fig. 11. Optical communication substrates improve large model training times by increasing inter-node bandwidth and enabling more accelerators per node for existing accelerator designs, and allowing increased intra-node bandwidth for future accelerator designs.

We explore these performance benefits for training a large GLaM model [39] on 3072 accelerators. We assume 8-bit precision and a batch size of 8192. TP is exploited within a node and DP across nodes. The accelerator is modeled after the Nvidia H100 GPU [10], with details shown in Table IV.

Fig. 11 shows the performance benefits. The first bar represents a reference implementation using 8 accelerators in a node communicating inside the node with NVlink interconnect and between nodes with 8 NDR InfiniBand network cards. The second bar shows an equal amount of accelerators in a node, but using *Opt. 1* to increase the inter-node bandwidth equal to the accelerator off-chip bandwidth multiplied by the number of accelerators on the edge of the substrate (8 for a 4x2 config). All communication overhead is heavily reduced, especially the overhead related to the MoE communication (reduced by a factor ~ 6). Only intra-node TP communication stays equal because the intra-node bandwidth was not changed. Overall, the *Opt. 1* leads to a 42% performance improvement in this scenario. The following bars (bars 3 to 5) show *Opt. 2*, where we use the increasing number of accelerators inside a node to exploit more tensor parallelism. We chose 4x4, 4x8, and 6x8 (estimated to be the max. rectangular configuration to put on a 300mm substrate when using near-reticle sized dies) configurations, resulting in 16, 32, and 48 accelerators in a node, with inter-node bandwidths equal to 12, 20, and 24 times the off-chip bandwidth of a single accelerator, respectively. This is because not all accelerators are on the edge of the communication substrate, so not all accelerators are connected with a dedicated fiber attachment. The performance increases because more TP is used compared to DP, so the effective minibatch size increases, hence the accelerators compute more efficiently. Overall, *Opt. 2* results in 29% higher performance when using 48 accelerators in a node compared to 8, on top of *Opt. 1*. The figure also shows the effect of *Opt. 3*: when using the advantage of the optical communication substrate, off-chip bandwidth can be increased for future accelerator designs. Bars 6 and 7 in Fig. 11 show how doubling and quadrupling the off-chip bandwidth increases system performance by 54% and 110%, respectively, for a system with 48 accelerators per node that already benefits from *Opt. 1* and *Opt. 2*. Note that computation time still remains the same, and starts to dominate training time for systems with high bandwidth.

These three optimizations together result in vast performance improvements for distributed deep learning training systems, up to almost $4\times$ the performance of the reference system, without increasing the peak computational ability of the accelerator.

IX. Conclusion

In this work, we proposed *AMPeD*, an analytical model for performance in distributed training of transformers. *AMPeD* provides the users with multiple tunable knobs, such as all the transformer model parameters, potential parallelism choices (along with their mapping onto the system), and the accelerator as well as system architecture specifications, thereby enabling hardware-software co-design. We demonstrated the capability of *AMPeD* by providing insights into the training time of transformers on current and future distributed system architectures with the help of 3 case studies. For example, we show how future distributed systems utilizing optical communication substrates pave the way to more efficient training of large models, with up to $4\times$ more performance than the current state-of-the-art systems without modifying the peak computational power of the accelerators. Finally, we validate the predictions from *AMPeD* with experiments on real systems and via published data, demonstrating a maximal error of 12%.

Furthermore, *AMPeD* can be easily extended for heterogeneous accelerators. Here, we incorporate the impact of memory constraint using the fitting function for microbatch efficiency. In the future, we plan to develop a comprehensive model to incorporate the memory constraints in the predictions.

REFERENCES

[1] G. V. Research. (2022) Deep learning market size,2022 - 2030. [Online]. Available: https://www.grandviewresearch.com/industry-analysis/deep-learning-market

[2] 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.

[3] Q. Wang, B. Li, T. Xiao, J. Zhu, C. Li, D. F. Wong, and L. S. Chao, "Learning deep transformer models for machine translation," *arXiv preprint arXiv:1906.01787*, 2019.

[4] U. Naseem, I. Razzak, K. Musial, and M. Imran, "Transformer based deep intelligent contextual embedding for twitter sentiment analysis," *Future Generation Computer Systems*, vol. 113, pp. 58–69, 2020.

[5] A. Gulati, J. Qin, C.-C. Chiu, N. Parmar, Y. Zhang, J. Yu, W. Han, S. Wang, Z. Zhang, Y. Wu *et al.*, "Conformer: Convolution-augmented transformer for speech recognition," *arXiv preprint arXiv:2005.08100*, 2020.

[6] Z. Shaheen, G. Wohlgenannt, and E. Filtz, "Large scale legal text classification using transformer models," *arXiv preprint arXiv:2010.12871*, 2020.

[7] A. Arnab, M. Dehghani, G. Heigold, C. Sun, M. Lučić, and C. Schmid, "Vivit: A video vision transformer," 2021. [Online]. Available: https://arxiv.org/abs/2103.15691

[8] D. Narayanan *et al.*, "Efficient large-scale language model training on gpu clusters using megatron-lm," in *SC*, 2021.

[9] X. Wang, Y. Xiong, X. Qian, Y. Wei, L. Li, and M. Wang, "Lightseq2: Accelerated training for transformer-based models on gpus," *arXiv preprint arXiv:2110.05722*, 2021.

[10] J. Choquette, "Nvidia hopper gpu: Scaling performance," in *2022 IEEE Hot Chips 34 Symposium (HCS)*, 2022, pp. 1–46.

[11] D. Moolchandani, A. Kumar, and S. R. Sarangi, "Performance and power prediction for concurrent execution on gpus," *ACM TACO*, vol. 19, no. 3, pp. 1–27, 2022.

[12] Y. Zhang, Z. Zheng, and M. R. Lyu, "Real-time performance prediction for cloud components," in *IEEE ISORCW*. IEEE, 2012.

[13] H. Bouzidi, H. Ouarnoughi, S. Niar, and A. A. E. Cadi, "Performance prediction for convolutional neural networks on edge gpus," in *ACM CF*, 2021.

[14] E. Gianniti, L. Zhang, and D. Ardagna, "Performance prediction of gpu-based deep learning applications," in *SBAC-PAD*. IEEE, 2018.

[15] S. Lym, D. Lee, M. O'Connor, N. Chatterjee, and M. Erez, "Delta: Gpu performance model for deep learning applications with in-depth memory system traffic analysis," in *IEEE ISPASS*, 2019.

[16] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," 2017. [Online]. Available: https://arxiv.org/abs/1706.03762

[17] S. Rajbhandari *et al.*, "Zero: Memory optimizations toward training trillion parameter models," in *SC*. IEEE, 2020.

[18] 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.

[19] F. Yan, O. Ruwase, Y. He, and T. Chilimbi, "Performance modeling and scalability optimization of distributed deep learning systems," in *ACM KDD*, 2015.

[20] H. Qi, E. R. Sparks, and A. Talwalkar, "Paleo: A performance model for deep neural networks," *OpenReview*, 2016.

[21] X. Y. Geoffrey, Y. Gao, P. Golikov, and G. Pekhimenko, "Habitat: A {Runtime-Based} computational performance predictor for deep neural network training," in *USENIX ATC 21*, 2021.

[22] S. Rashidi, S. Sridharan, S. Srinivasan, and T. Krishna, "Astra-sim: Enabling sw/hw co-design exploration for distributed dl training platforms," in *IEEE ISPASS*, 2020.

[23] Nvidia. (2022) Optimizing linear/fully-connected layers. [Online]. Available: https://docs.nvidia.com/deeplearning/performance/pdf/Optimizing-Linear-Fully-Connected-Layers-User-Guide.pdf

[24] N. S. Keskar, D. Mudigere, J. Nocedal, M. Smelyanskiy, and P. T. P. Tang, "On large-batch training for deep learning: Generalization gap and sharp minima," *arXiv preprint arXiv:1609.04836*, 2018.

[25] M. Yu, Y. Tian, B. Ji, C. Wu, H. Rajan, and J. Liu, "Gadget: Online resource optimization for scheduling ring-all-reduce learning jobs," *arXiv preprint arXiv:2202.01158*, 2022.

[26] 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," *arXiv preprint arXiv:1811.06965*, 2018.

[27] A. Karpathy, "mingpt, a pytorch re-implementation of gpt, both training and inference." 2022. [Online]. Available: https://github.com/karpathy/minGPT

[28] C. Kim, H. Lee, M. Jeong, W. Baek, B. Yoon, I. Kim, S. Lim, and S. Kim, "torchgpipe: On-the-fly pipeline parallelism for training giant models," 2020. [Online]. Available: https://arxiv.org/abs/2004.09910

[29] T. Wolf, L. Debut, V. Sanh, J. Chaumond, C. Delangue, A. Moi, P. Cistac, T. Rault, R. Louf, M. Funtowicz *et al.*, "Huggingface's transformers: State-of-the-art natural language processing," *arXiv preprint arXiv:1910.03771*, 2019.

[30] J. He, J. Qiu, A. Zeng, Z. Yang, J. Zhai, and J. Tang, "Fastmoe: A fast mixture-of-expert training system," *arXiv preprint arXiv:2103.13262*, 2021.

[31] Z. Ma *et al.*, "Bagualu: targeting brain scale pretrained models with over 37 million cores," in *PPoPP*, 2022.

[32] Nvidia. (2020) Nvidia a100 tensor core gpu architecture. [Online]. Available: https://images.nvidia.com/aem-dam/en-zz/Solutions/data-center/nvidia-ampere-architecture-whitepaper.pdf

[33] N. S. Keskar, D. Mudigere, J. Nocedal, M. Smelyanskiy, and P. T. P. Tang, "On large-batch training for deep learning: Generalization gap and sharp minima," *arXiv preprint arXiv:1609.04836*, 2016.

[34] P. Goyal, P. Dollár, R. Girshick, P. Noordhuis, L. Wesolowski, A. Kyrola, A. Tulloch, Y. Jia, and K. He, "Accurate, large minibatch sgd: Training imagenet in 1 hour," *arXiv preprint arXiv:1706.02677*, 2017.

[35] A. Krizhevsky, "One weird trick for parallelizing convolutional neural networks," *arXiv preprint arXiv:1404.5997*, 2014.

[36] N. C. Harris, D. Bunandar, A. Joshi, A. Basumallik, and R. Turner, "Passage: A wafer-scale programmable photonic communication substrate," in *2022 IEEE Hot Chips 34 Symposium (HCS)*, 2022, pp. 1–26.

[37] J. Van Campenhout, "Silicon photonics technology for terabit-scale optical i/o (invited)," in *2021 ACM/IEEE SLIP*, 2021.

[38] M. Moralis-Pegios, S. Pitris, C. Mitsolidou, K. Fotiadis, H. Ramon, J. Lambrecht, J. Bauwelinck, X. Yin, Y. Ban, P. De Heyn *et al.*, "Silicon circuits for chip-to-chip communications in multi-socket server board interconnects," *IET Optoelectronics*, vol. 15, no. 2, pp. 102–110, 2021.

[39] N. Du, Y. Huang, A. M. Dai, S. Tong, D. Lepikhin, Y. Xu, M. Krikun, Y. Zhou, A. W. Yu, O. Firat *et al.*, "Glam: Efficient scaling of language models with mixture-of-experts," in *ICML*, 2022.