Measuring the Impact of Gradient Accumulation on Cloud-based Distributed Training

Zimeng Huang*, Bo Jiang*, Tian Guo†, Yunzhuo Liu*
*Shanghai Jiao Tong University, China
†Worcester Polytechnic Institute, U.S
*{lukehuang, bjiang, liu445126256}@sjtu.edu.cn, †tian@wpi.edu

***Abstract*—Gradient accumulation (GA) is a commonly adopted technique for addressing the GPU memory shortage problem in model training. It reduces memory consumption at the cost of increased computation time. Although widely used, its benefits to model training have not been systematically studied. Our work evaluates and summarizes the benefits of GA, especially in cloud-based distributed training scenarios, where training cost is determined by both execution time and resource consumption. We focus on how GA can be utilized to balance execution time and resource consumption to achieve the lowest bills. Through empirical evaluations on AliCloud platforms, we observe that the total training cost can be reduced by 31.2% on average with a 17.3% increase in training time, when GA is introduced in the large-model and small-bandwidth scenarios with data-parallel training strategies. Besides, taking micro-batch size into optimization can further decrease training time and cost by 21.2% and 24.8% on average, respectively, for hybrid-parallel strategies in large-model and GPU training scenarios.**

***Index Terms*—gradient accumulation, distributed training, cloud computing**

I. Introduction

Gradient accumulation (GA) [1], [2] is a technique commonly used in deep learning (DL) to train large models that require high memory resources. GA divides the training data into smaller micro-batches and then accumulates the gradients from each micro-batch before applying them to update the model. GA has been used to address the GPU mismatch between model training requirement and the GPU memory [3]. However, prior work also demonstrates the possibility of prolonged training time when using GA [4], [5], suggesting the need to investigate the trade-offs between training time and GPU memory in practice.

In this paper, we are interested in answering the key question of *what distributed training scenarios can benefit from GA and by how much*. In particular, our study focuses on cloud-based distributed training due to its popularity and its vast resource options that DL practitioners can explore to speed up training. For example, GPU-equipped VMs offered by Infrastructure-as-a-Service (IaaS) [6]–[8] have been widely used to train DL models and many prior works have investigated improving cloud-based training performance by addressing the communication bottlenecks [4], [5], [9] or utilizing cheaper preemptible resources [10]. However, there still lacks a systematic study of the impact of GA, a promising approach to improve the computation-to-communication ratio, on IaaS-based training.

Furthermore, we have also observed an increased interest in leveraging Function-as-a-Service (FaaS) workers [11]–[13] to train DL models [14], [15], due to FaaS' key benefits in true pay-as-you-go pricing model and the ability to scale to many workers. These FaaS-based training frameworks are still in their infancy and could benefit from further optimization opportunities associated with GA. In short, our study evaluates the impact of GA on two popular cloud resource offerings: VMs by IaaS and serverless functions by FaaS.

Another dimension, besides the cloud resource offerings, that can also impact the distributed training performance is the parallelism strategy. Commonly implemented strategies include data parallelism (DP) and hybrid parallelism (HP) [5], [9], which differ in how the training workload is divided and scheduled. In this work, we consider both DP and HP due to their popularity. We define the combination of the cloud resource offerings and parallelism strategy as an instance of *training scenario*. For each scenario, we choose a representative training framework and measure the training performance (time, cost, and accuracy), both with and without integration with GA.

When training with DP [5], the training dataset is partitioned and each *worker*, i.e., training node, is assigned a subset. Each worker has a copy of the complete model, trains with their own data per batch, and performs gradient synchronization with each other to update model parameters. In essence, DP allows training models with larger global batch size by having more workers [16] , each in charge of a *local batch*, a proportion of the global batch. However, more workers also mean larger synchronization overhead which often translates to inferior training performance.

GA brings the opportunity to alleviate the abovementioned synchronization overhead because it decouples the configuration of the number of workers from the global batch size. That is, unlike vanilla DP-based training, we can use any number of workers for a given DL model and global batch size when using GA with DP-based training. Moreover, GA also softens the requirement on GPU memory, therefore increasing the pool of eligible cloud configurations. To better leverage GA to improve training performance, we need to address the added resource configuration problem, e.g., how many workers to use and the worker resource configuration. In this work, we use

The corresponding author is Bo Jiang.

Bayesian Optimization (BO) to find the desired configuration.

Training with HP means that both data parallelism and model parallelism are used. In this paper, we focus on a specific variant of model parallelism called *pipeline parallelism* due to the more available framework support. Pipeline parallelism divides a DL model into multiple stages, each consisting of a number of layers, and runs each stage on one or more workers depending on whether DP is also used. During training, each input batch is further divided into multiple micro-batches to go through different stages in a pipelined fashion. A typical DNN training iteration contains a forward pass and a backward pass through all stages. In each phase, different micro-batches are calculated successively in the forward pass and the activations generated are stored for reuse in the backward pass. Note GA is naturally incorporated in HP through the use of micro-batches in the pipeline.

Our empirical observation shows that HP-based training performance can be affected by micro-batch size (*mbs*). However, many existing works [9], [17] manually set the *mbs* for a given setup, lacking details on its impact or how to configure it for different training scenarios. Moreover, we observe that different *mbs* will affect the time to complete a full batch training. Therefore, for HP-based training, we will consider configurations of *mbs* when measuring the impact of GA.

In evaluating GA's impact on cloud-based distributed training, we make the following key contributions.

- **We establish the potential of GA in improving cloud-based distributed training via empirical measurement and performance modeling.** For DP-based scenarios, we find that directly applying GA can further optimize the training cost via adjusting the number of workers; for HP-based scenarios, we find that considering *mbs* in the optimization problem can regulate the computation time of one full iteration to achieve better training time and cost.
- **Our evaluation with four frameworks on a public cloud shows that GA can help further reduce training time and cost.** For example, we find that GA can be effectively combined with DP for training large models with low network bandwidth. For some training scenarios, the combination of GA and DP can reduce the training cost by 31.2% while increasing the total training time by 17.3% correspondingly, whereas considering the micro-batch size optimization in the HP scenario can reduce the training time by 21.2% and the cost by 24.8% on average.

The rest of the paper is organized as follows: §II introduces common cloud-based training scenarios and the gradient accumulation technique, in addition to reviewing the related work in distributed training techniques. §III identifies the potential opportunities and optimization effects of GA in cloud-based distributed training scenarios through preliminary empirical evaluation and analytical modeling. §IV elaborates on the evaluation methodology and corresponding design details we employ. §V presents the evaluation results of GA with various distributed training frameworks regarding training cost and speed. §VI concludes this paper and identifies possible future work on the training framework design.

II. Background and Related Work

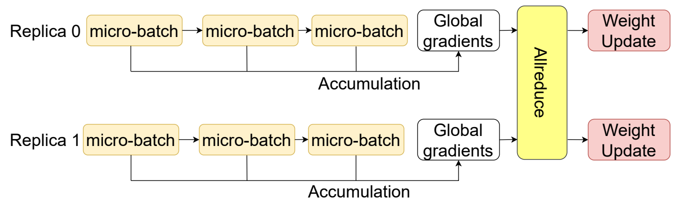
In this section, we provide the background of common *cloud-based training scenarios*, where each scenario describes the cloud resource offering (§II-A) and training parallelism strategy (§II-B). We also explain the integration of *gradient accumulation* in different scenarios in detail in §II-C.

A. Cloud Resource Offerings

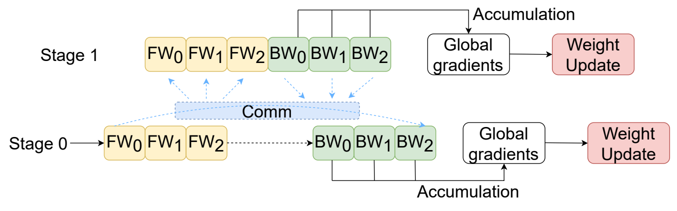
Popular cloud providers such as AWS and Azure offer computation resources to cloud customers in two main ways: infrastructure-as-a-service (IaaS) and function-as-a-service (FaaS). The former has become the de facto way for deep learning practitioners to perform *distributed training*, which often involves training a deep learning model using network-connected workers equipped with GPU devices [7], [8]. The latter, FaaS, due to its attractive benefits such as true pay-as-you-go, has attracted many research interests to train deep learning models with it [14], [15], [18]. Next, we briefly introduce the key characteristics of IaaS and FaaS, which will serve as the basis for understanding vastly different deep learning training framework designs.

IaaS is the most commonly used cloud computing service for deep learning [19], where resources such as servers, storage, and networks are provided directly to users in the form of rental, such as virtual machine (VM) rental service. IaaS is the simplest cloud computing delivery mode, it provides cloud resources to users by virtualization techniques. However, tenants still need to configure the underlying server settings and the corresponding computing environment. Many cloud service platforms nowadays provide a variety of virtual machine families for different computing scenarios, which makes it possible to tune different resource configurations. At present, some prior work has pointed out the possibility of resource configuration tuning in IaaS [18], [20], [21].

FaaS is also being used by more and more developers in recent years. The design idea of FaaS is to divide a single computing operation that interacts with the server into independent functions. The developer can directly deploy and run the service code in stateless computing containers provided by the cloud service provider. The developer only needs to write the function code of the business logic without worrying about server resource management. Although FaaS was originally developed for web microservices and Internet of Things (IoT) applications, recently researchers have also explored the serverless service's role in DNN training [14], [15], [18]. Due to serverless applying a more fine-grained charging strategy and strong resource elasticity, it is also worth looking forward to using serverless service for DNN distributed training in the future. However, the inability to direct communication between FaaS nodes and the low bandwidth characteristics are worthy of attention when we design the training framework. Besides, the present-day commercial FaaS does not support GPU, which is also a key difference between IaaS and FaaS.



(a) GA in Data Parallelism



(b) GA in Pipeline Parallelism

Fig. 1: **Gradient accumulation in different parallelism.** For data parallelism, the use of GA splits the running batch into micro-batches and accumulates the corresponding gradient in each worker. After the Allreduce operation synchronizes gradients among all replicas, each worker updates their local weights. For pipeline parallelism, each stage produces and accumulates gradients for all micro-batches, although the forward and backward processes are separated. We split the global batch into three micro-batches as an example.

B. Distributed Training Parallelism

Distributed training parallelism defines how the training workload of deep learning is distributed among different workers. In this section, we describe the two parallelism strategies, *data parallelism* and *hybrid parallelism*, that are supported by existing training frameworks from industry and academia [5], [9], [14]. These two parallelism strategies exhibit significant differences in the training process and need to be integrated with gradient accumulation differently, as described in §II-C.

Data parallelism (DP) is the most widely adopted parallelism strategy [4], [5], [22], where each worker saves a replica of the entire model and part of the dataset. In a training iteration, workers compute the gradient of their local data in parallel, then exchange the gradients to update the model parameters. Data parallelism is easy to implement, but its training efficiency is often limited by the dependency between computation and communication. There is a lot of work on how to overlap computing and communication processes under data parallel design. For example, the DDP framework of Pytorch [5] effectively reduces the total training time by overlapping the gradient computation and transmission during backpropagation, while LAGA [4] explores the asynchronous overlap method with the utilization of GA.

Hybrid parallelism (HP) combines both data parallelism and *model parallelism*, which divides the DNN model between workers to reduce the memory bottleneck of distributed training [23]–[25]. While there are many variants of model parallelism [26], [27], a specific form called *pipeline parallelism* has become the most common variant [9], [14], [17], [28]. In pipeline parallelism, each worker retains part of the model layers and splits the input data so that the entire training process can be pipelined. Numerous frameworks, including Gpipe [17] and DAPPLE [9], have implemented pipeline parallelism to improve overall training throughout. For example, Gpipe [17] explores the large model training method of *synchronous pipeline approach*, where each data batch is divided into several micro-batches at the beginning of each iteration, and each worker respectively carries out forward processing (FW) for each micro-batch and saves the activation value. The gradient is accumulated in the backward processing (BW), and the model parameters are updated. DAPPLE [9] optimizes the memory consumption in the synchronous pipeline method, further reducing the memory overhead in the training process. Recent frameworks [9], [14] further support pipeline parallelism in the context of hybrid parallelism. For the remainder of the paper, we will evaluate the impact of GA on hybrid parallelism, which includes pipeline parallelism as a special case and has a larger search space of distributed training configurations. Note that these configurations include the number and types of workers, the model splitting schemes, and the micro-batch size used for training.

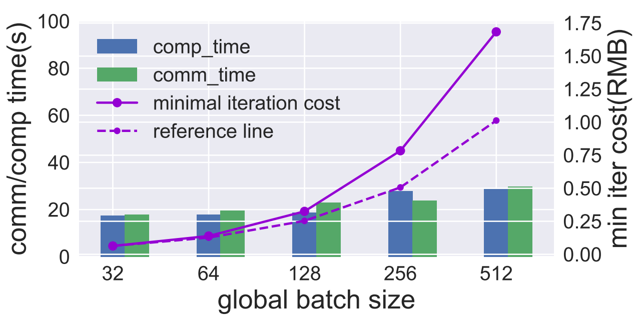
C. Gradient Accumulation

Gradient accumulation (GA) is a technique that was proposed to address the GPU memory shortage problem by the machine learning community [1], [2]. In this paper, we are interested in exploring the effectiveness of GA under different cloud training scenarios.

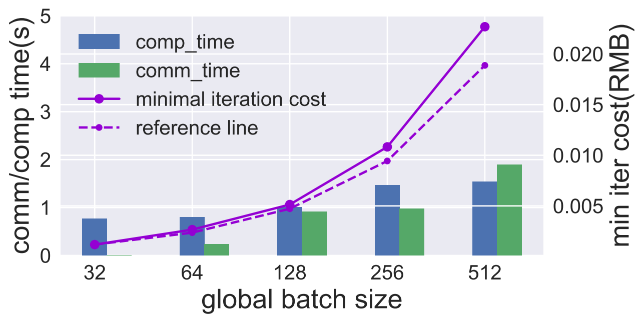
The basic idea of gradient accumulation is to divide the training batch in each iteration into multiple micro-batches. After a micro-batch completes the backward propagation, we do not immediately update the model but accumulate the micro-batch gradients to obtain the global batch gradient. We update the model only when all micro-batches in the current batch have been processed; see Figure 1. In this process, the splits for each training batch are called gradient accumulation steps (GA steps). For the convenience of further discussion, we refer to the overall training batch size as global batch size (*gbs*), and the split batch size as micro-batch size (*mbs*). The combination of GA and different parallelisms is not the same due to the differences in the training process:

In data parallelism (Figure 1a), GA can reduce the maximum memory consumption of a worker with a smaller *mbs*, but the overall training time will increase correspondingly. By using different *mbs*, GA can provide a tradeoff between memory consumption and training time.

Most existing frameworks with synchronous pipeline parallelism utilize GA in their design [9], [17], [29]. Specifically, a global batch is divided into multiple micro-batches,



(a) FaaS



(b) IaaS

Fig. 2: **The minimal cost and corresponding communication/computation time for one iteration without using GA.** We use *Amoebanet-D18* model and *Cifar-10* dataset in this experiment. The dotted line in the figure represents the reference line assuming that minimal iteration cost is proportional to *gbs*.

and the gradient of a micro-batch is accumulated when the corresponding backward calculations are completed. In this process, adjusting the size of *mbs* will also affect the overall performance of training, as discussed in detail in §III.

It is worth noting that the use of GA can also affect the model's accuracy, especially for layers that need to be calculated across micro-batches, such as Batch Normalization (BN) layers [30]. The direct use of GA may affect the parameter learning of these layers [17], thus affecting the final performance of the model. Taking the BN layer as an example, in the process of gradient calculation, we need to calculate the statistics of the whole batch data, but when a batch is split into multiple micro-batches, only the data in the micro-batch can be counted in the forward and backward calculation, which results in a certain negative impact on the parameter learning of this layer. In this paper, we also discuss the influence of GA on the accuracy of the model and evaluate the degree of influence in §V-B.

III. Identifying Opportunity of GA via Empirical Measurement and Performance Modeling

In theory, GA is a promising technique to improve cloud training cost and time. However, it is unclear whether different cloud training scenarios can benefit from GA in practice. In this section, we first demonstrate the potential for improvement in four different cloud training scenarios (DP-IaaS, DP-FaaS, HP-IaaS, HP-FaaS) via an empirical measurement conducted on AliCloud, a popular cloud provider in China. For each scenario, we further use training performance models to analyze the time and cost improvement *when integrated with GA*. In short, our analyses pinpoint that GA can be an effective technique in practice.

A. Opportunities for Data Parallelism

Although many existing works have given corresponding resource configuration optimization methods for DP, these studies have not considered the resource optimization of DP combined with GA. In fact, GA provides the ability to adjust the number of workers and may further reduce the training cost in the DP scenario, making it a valuable addition.

For DP, the duration of a single iteration is primarily affected by the computation time and the communication time for model synchronization. Given that the resources available on a single worker node are limited (e.g. up to 80GB for VM (IaaS) and 32GB for serverless (FaaS) in AliCloud [6], [11]), scaling up the number of workers becomes necessary as *gbs* increases, resulting in a noticeable hike in the overall cost caused by the increasing communication time and the number of invested instances. Our experiments on the AliCloud platform (Figure 2) demonstrate that the optimal cost of training one iteration of *Amoebanet-D18* model increases superlinearly in the *gbs*, which suggests the potential for cost optimization in data parallel training. This phenomenon of superlinear growth stems from two distinct factors. First, as the scale of training expands, communication time increases with the number of nodes involved. Second, for varying workloads, the optimization process often yields dissimilar optimal configurations. Additionally, the correlation between instance performance and unit price of cloud service providers is frequently nonlinear. By comparing different training scenarios, it can be distinctly observed that the growth of minimal iteration cost is more significant with the increasing *gbs* in FaaS which has limited bandwidth and cheaper instances. In contrast, training *Amoebanet-D18* models in IaaS has alleviated this phenomenon.

The GA approach extends a new dimension to each data parallel worker node by allowing the accumulation of gradients from multiple micro-batches within a single node, thus reducing the overall number of workers. Additionally, GA also expands the range of resource configurations that can be used. By setting a small *mbs* value, we can choose a smaller resource configuration to complete the training process. We let m denote the micro-batch size used for DP training, n the num of worker nodes, p the price per unit time, s the number of GA steps, $T_{calc}(m)$ the computation time with micro-batch size m, $S_{over}(m)$ the overlapped part of communication and

TABLE I: **Total computation time and iteration time with different *mbs* for training *Resnet101*.**

Micro-batch size	T_{comp}(s) for HP-FaaS	T_{comp}(s) for HP-IaaS (T4)	T_{iter}(s) for HP-FaaS	T_{iter}(s) for HP-IaaS (T4)
1	182.41	16.21	58.33	5.53
2	168.74	10.42	54.92	3.58
4	161.37	8.74	52.73	3.17
8	155.82	5.88	51.65	3.24
16	OOM	5.24	OOM	2.76
32	OOM	5.13	OOM	2.82

computation, and $S_{all}(n)$ the total communication time. If we do not use GA (which means $s = 1$), the training time and training cost for one iteration can be represented as:

$$T_{iter} = T_{calc}(m) + S_{all}(n) - S_{over}(m)$$
$$C_{iter} = np \times [T_{calc}(m) + S_{all}(n) - S_{over}(m)]$$

Keeping the other configurations unchanged, we can reduce the number of workers by using GA. In the above example, suppose we double the number of GA steps ($s' = 2$), the number of worker nodes can be halved ($n' = \frac{n}{2}$). The training time and training cost are then changed as:

$$\begin{aligned} T'_{iter} &= 2 \times T_{calc}(m) + S_{all}(n/2) - S_{over}(m) \\ C'_{iter} &= \frac{np}{2} \times [2 \times T_{calc}(m) + S_{all}(\frac{n}{2}) - S_{over}(m)] \\ &= np \times [T_{calc}(m) + \frac{1}{2} S_{all}(\frac{n}{2}) - \frac{1}{2} S_{over}(m)] \end{aligned}$$

Note that $C'_{iter} < C_{iter}$, which shows that employing GA in the current resource configuration schemes could reduce the training cost. Nevertheless, it is worth noting that when the number of workers is small and the value of *mbs* is high, the training time of an iteration may increase correspondingly.

B. Opportunities for Hybrid Parallelism

At present, most of the existing hybrid parallel training frameworks set *mbs* as a hyperparameter that requires manual setting, without accounting for how *mbs* should vary with different training workloads [9], [14], [17], this is equivalent to disregarding GA's tradeoff effect on memory usage and training time. To demonstrate the impact of *mbs* on the overall computation time, we conduct an experiment involving the training of *Resnet101* under the FuncPipe framework with three pipeline stages, consisting of 119, 134, and 51 model layers respectively. We keep *gbs* fixed at 256 and vary the value of *mbs*. The experiments are performed on AliCloud serverless 3GB instances (FaaS) and gn6i.xlarge instances (IaaS with an NVIDIA T4 GPU), and the number of instances used is determined by the FuncPipe analytical optimizer. For each iteration, we measure the sum of the computation time of all three stages, denoted by T_{comp}, and the actual training time, denoted by T_{iter}. The results are presented in Table I. It is worth noting that T_{iter} is significantly shorter than T_{comp}, as computations of different stages overlap with each other.

We observe that *mbs* has a non-negligible effect on both the total computation time T_{comp} and the iteration time T_{iter}. As

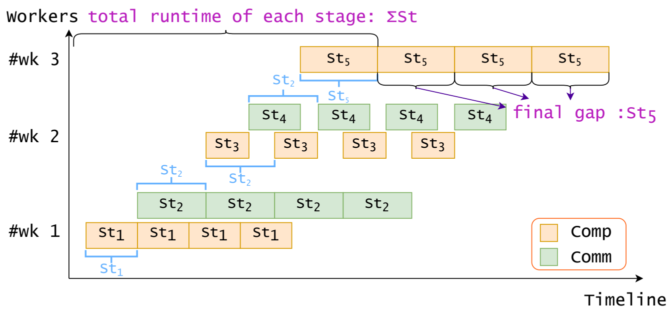


Fig. 3: **Analysis of FuncPipe's forward process.** During pipeline's each stage, the input gap between adjacent micro-batch tasks can be stretched by larger stage running time. The analysis is similar to the backward process.

mbs varies from 1 to 32, T_{comp} can be reduced by 18.70% for FaaS and 68.35% for IaaS, while T_{iter} can be reduced by 11.45% for FaaS and 49.01% for IaaS. These results highlight the potential benefits of considering *mbs* as an optimization variable to further enhance the efficiency of training, both in terms of time and cost. However, it is important to note that the overall computation time T_{comp} and the iteration time T_{iter} cannot continue to decrease indefinitely, since a larger value of *mbs* corresponds to greater memory consumption by a single worker node, which may eventually result in Out-Of-Memory (OOM) errors. In addition, while the overall computation time T_{comp} decreases monotonically with increasing *mbs*, the iteration time T_{iter} does not decrease monotonically. To better understand the relationship between *mbs* and T_{iter}, we conduct a detailed analysis below.

In pipeline parallelism, we differentiate computation tasks and communication tasks as distinct stages when constructing the pipeline. To describe the running time of each stage, we use the notation $St_i(m)$, where i denotes the stage ID and m represents the value of *mbs*, as illustrated in Figure 3. We can divide the overall iteration time into two parts: the total running time of each stage for one *mbs* and the final gap between two successive micro-batches. Note that the final gap between two successive micro-batches is determined by the maximum stage running time. The backward pass is similar to the forward pass, with the initial gap between successive micro-batches given by the final gap in the forward pass. It should be noted that we assign stage IDs to both the forward and backward processes. Assuming the partition of model stages is fixed with total s stages and *gbs* = B, we can express the training time for a full iteration in FuncPipe as follows:

$$T_{iter} = \sum_{k=1}^{s} St_k(m) + (\frac{B}{m} - 1) \max_k St_k(m) \qquad (1)$$

Prior work [14], [31] have established that the time of a single stage St_i exhibits a linear growth pattern with respect to *mbs* and the relationship between $St_i(m)$ and *mbs* can be represented as follows:

$$St_i(m) = \alpha_i \cdot m + \beta_i$$

where α_i and β_i are the coefficients in the i-th stage. Thus the total iteration time in (1) is given by

$$T_{iter} = \sum_{k=1}^{s}(\alpha_k m + \beta_k) + (\frac{B}{m} - 1) \cdot (\alpha^* m + \beta^*)$$
$$= B\alpha^* + (\sum_{k=1}^{s}\beta_k - \beta^*) + m(\sum_{k=1}^{s}\alpha_k - \alpha^*) + \frac{B}{m}\beta^*, \quad (2)$$

where α^* and β^* correspond to the coefficients of the slowest stage. Note that adjusting the *mbs* can optimize the total iteration time by altering the $m(\sum_{k=1}^{s}\alpha_k - \alpha^*)$ and $\frac{B}{m}\beta^*$ part above, which also theoretically explains why tuning the *mbs* results in a reduced iteration time.

It is worth noting that DAPPLE and FuncPipe differ significantly in terms of their training process design. Specifically, in DAPPLE (HP-IaaS), the pipeline is configured to adapt the order of task execution, with training being performed in a one-forward-one-backward (1F1B) order in each stage [9]. Within the DAPPLE framework, when the training process stabilizes, all GPUs in the pipeline are fully utilized, resulting in the total training time being predominantly influenced by the running time of the slowest stage. Considering the warmup and stable phases of training, the overall iteration time for DAPPLE can be expressed as:

$$T_{iter} = \frac{B}{m}(\alpha_1 m + \beta_1) + (\frac{B}{m} - 1) \cdot (\alpha^* m + \beta^*)$$
$$= B\alpha_1 + B\alpha^* - \beta^* + \frac{B}{m}(\beta_1 + \beta^*) - \alpha^* m \quad (3)$$

The difference between (2) and (3) suggests that the optimal *mbs* values may vary for different training frameworks in distinct scenarios. Additionally, it is worth noting that the changes in the time coefficient α_i attributable to modifications in configurations tend to have a more pronounced effect in the IaaS scenario than β_i, which will also cause an impact on the optimal *mbs* value in different scenarios. The impact of GA on HP-based training scenarios will be evaluated in §V.

It should be noted that the analytical models for C_{iter} and T_{iter} in this section are for fixed model splitting schemes. In the HP scenario, where the model splitting scheme can vary, it is unclear yet how to model C_{iter} and T_{iter} for DAPPLE.

IV. Evaluation Methodology

In this section, we present our evaluation methodology for answering the key question of *the impact of GA on cloud-based distributed training*. Specifically, we design experiments for four cloud-based training scenarios to quantify GA's benefits, in terms of training time and monetary cost.

A. Training Workload

To cover different types of deep learning training workloads, we choose three models: ResNet [32], BERT [33], and AmoebaNet [34]. ResNet is a widely used CNN model, BERT a Transformer-based model, and AmoebaNet a model based on the NASNet structure. To illustrate the training cost optimization effect of different model sizes, we select Resnet101, Amoebanet-D18, Amoebanet-D36, and BERT-Large for experiments, whose size ranges from 170MB to 1GB. Table II shows the sizes of the different models and the activation values in detail. We use the popular image classification dataset *Cifar-10* to train Resnet101, Amoebanet-D18, and Amoebanet-D36. As for the BERT-Large model, we use the *SQuAD* benchmark [35] to run the fine-tuning task.

TABLE II: **Training models used in evaluation**

Model name	**Parameter size (MB)**	**Activation size per sample(MB)**
Resnet101	170	198
Amoebanet-18	476	432
Amoebanet-36	900	697
Bert-Large	1153	263

B. Available Cloud Resource Configurations

For all our experiments, we use a public cloud provider Alibaba Cloud, which provides both IaaS and FaaS services[1]. For the serverless service, we can control the resources of the function instances by selecting specific function memory sizes (accurate to MB). For VM instances, we need to choose between various configurations offered by the cloud service provider. Specifically, we choose eight discrete memory sizes (1024MB, 1536MB, 2048MB, 2560MB, 3072MB, 4096MB, 8192MB, and 16384MB) on AliCloud Function Computing Platform [11] for FaaS, and six families on Alibaba Cloud ECS [6] for IaaS: gn6i, gn7i (general purpose with GPU), vgn7i-vws (light weight GPU), sgn7i (sharing type GPU), gn6v (V100 GPU) and gn7e (A100 GPU). In this paper, we only focus on single-GPU distributed training[2]. We show all the VM instances used in the experiments in Table III. Our current selection of instance types is limited by time and budget constraints. Since sharing-type instances adopt non-binding resource scheduling mode and their performance may be affected by multi-tenant environments, we repeat each experiment 5 times and take the average of the measurements as the final test result when evaluating with sharing-type instances.

C. Evaluation Frameworks

Considering the combination of different cloud resource offerings and parallelism strategies, we explore four different cloud-based distributed training scenarios, as shown in Table IV. For each scenario, we choose a representative training framework for carrying out experiments. For IaaS-based scenarios with GPU-equipped VMs, DistributedDataParallel (DDP) [5] module proposed by PyTorch and DAPPLE [9] training framework are selected for testing. For FaaS-based scenarios with function-based serverless workers, LambdaML [15] and FuncPipe [14] frameworks are selected for

[1]Our experiment designs are not cloud-specific. Other cloud providers such as AWS and Google Cloud can also be used and should lead to similar results.

[2]Multi-GPU distributed training is also a common setup, but exhibits different communication patterns, which we will leave as future work.

TABLE III: **Configurations of VM instances used in the evaluation.** GPUs are all from NVIDIA. Note that the number of GPUs may be represented as a fraction, with the denominator indicating the number of tenants sharing a single GPU.

Instance name	**# vCPU**	**Memory (GB)**	**GPU**	**GPU memory (GB)**	**Bandwidth (Gbps)**	**Price (RMB/hour)**
gn6i.xlarge	4	15	1*T4	16	4	11.67
gn6i.2xlarge	8	31	1*T4	16	5	14.04
gn7i.2xlarge	8	30	1*A10	24	16	12.75
gn7i.4xlarge	16	60	1*A10	24	16	13.50
vgn7i-vws.xlarge	4	30	1*A10 * 1/6	4	3	3.117
vgn7i-vws.2xlarge	10	62	1*A10 * 1/3	8	5	5.609
vgn7i-vws.3xlarge	14	93	1*A10 * 1/2	12	8	8.102
sgn7i.xlarge	4	8	1*A10 * 1/12	2	5	1.875
sgn7i.2xlarge	8	16	1*A10 * 1/6	4	10	3.124
sgn7i.4xlarge	16	32	1*A10 * 1/3	8	20	5.621
gn6v.2xlarge	8	32	1*V100	16	2.5	26.46
gn7e.4xlarge	16	125	1*A100	80	25	34.742

TABLE IV: **Frameworks used for different cloud-based training scenarios.**

Training scenario	**Training framework**
IaaS+DP	DDP [5]
IaaS+HP	DAPPLE [9]
FaaS+DP	LambdaML [15]
FaaS+HP	FuncPipe [14]

the corresponding experiments. Furthermore, we ensure that the synchronization and communication architecture of various training frameworks remains consistent. Specifically, all the aforementioned training frameworks employ synchronous communication and rely on object storage service (OSS) [36] for intermediate storage and node communication.

Moreover, applying GA in distributed training scenarios can bring a new dimension to the overall training optimization process, effectively expanding the search space of the original optimization problem. To explicitly demonstrate the impact of GA on the optimization problem, we conduct experiments that compare the optimization results of using GA against those obtained without using GA, which we respectively denote as *"with GA"* and *"w/o GA"*. When we refer to the "with GA" settings, we explicitly exclude the corresponding "w/o GA" settings from consideration in the optimization. This helps us to better understand the impact of using GA on the training process and highlights the differences between the two approaches. Additionally, it is worth noting that GA is a training technique that is not specific to any particular framework and can be easily applied to different training frameworks. However, due to the significant differences between various parallelism strategies, the specific settings of GA may vary across different scenarios:

In the DP scenarios, the setting "w/o GA" represents the original settings when we use corresponding DP training frameworks with different cloud services, while "with GA" requires GA steps not equal to 1 in the training process.

As for HP experiments, we focus on the variation of *mbs* for different training scenarios. In the "w/o GA" situation, we use the default *mbs* setting of most pipeline parallelism frameworks (i.e., *mbs* = 4) for optimal training. In the "with GA" situation, we treat *mbs* as an optimization variable and present the optimized cost-time Pareto frontier under different weight settings in the objective function.

D. Resource Optimization Algorithm

To fairly compare the optimization effect of GA under different training scenarios, we employ the same optimization algorithm for determining the training resource configuration. Further, to simplify the resource optimization process, we utilize Bayesian Optimization to conduct black-box optimization for all training scenarios.

Problem formulation. In cloud-based training scenarios, both training time and training cost are important factors that users typically care about. Therefore, when optimizing for the overall training performance, we need to consider both time and cost dimensions. Here we use a simple weighted sum method to integrate these two different optimization objectives:

$$\min \quad C_{iter} + w \cdot T_{iter}.$$

Here, C_{iter} and T_{iter} represent one iteration's cost and time, respectively; w represents the weight value that controls the importance of time in the objective function. Considering different training workloads, the specific inputs for optimization are the DL model, the dataset, and the *gbs* chosen by users. In the following experiments, we choose C_{iter} and T_{iter} as the main performance metrics and adopt four different weight settings: $w = [0, 2^8, 2^{12}, 2^{16}]$.

Choosing optimization algorithm. There are three popular types of resource optimization algorithms nowadays, which are based on analytical modeling, Bayesian optimization (BO), and reinforcement learning (RL). State-of-the-art approaches employ analytical modeling for finding the optimal resource configuration [14], [21]. However, such methods must be customized for specific training frameworks. For some frameworks such as DAPPLE, it is unclear if there are analytical models that can be easily incorporated here. In addition, the same resource configuration can perform differently in the cloud environment. For example, in the multi-tenant shared cloud environment, stragglers may occur [20]. Therefore, compared with analytical models, BO also has the advantage of dynamic optimization in changeable cloud environments. On the other hand, prior work shows that RL-based approaches often have high additional training costs but can only bring relatively limited performance improvement [18]. Therefore, in this paper, we use BO to optimize the selection of resources.

BO [37] is a framework for solving optimization problems where we can observe the objective function through experiments. Starting from a few randomly selected sample points in the search space, BO profiles the actual results of these samples and estimates the value of the objective function by modeling the objective function as a stochastic process and computing the confidence interval. After one profiling step, the estimated objective function and the confidence interval will be updated correspondingly. In our experiments, we set BO to stop after 100 iterations.

To better use BO to find a reasonable configuration scheme, we must also make corresponding design decisions for BO: *Prior function.* Like most BO frameworks [20], [21], we choose the Gaussian process as our prior function. We also assume that the objective function is a sample of a Gaussian process. In different scenarios, the Gaussian process shows good flexibility and tractability. *Acquisition function.* We use the Expected Improvement (EI) [38] as our acquisition function for its ease of use and convenience. We maintain the consistency with HeterBO [21] in terms of the design details but discard the original constraints part.

Lastly, we use the idea of HeterBO [21] which utilizes the MLaaS training specific prior to limit the Bayesian Optimization search process: based on actual observations, the trend of scale-out speedup follows a concave-shaped curve, which means that we can early stop the search in the expensive scale-out region if we detect a decline in training speed between two neighboring deployments.

V. Evaluation Results

In this section, we evaluate and compare the optimization impact of GA in different cloud computing scenarios, and give corresponding analysis against the results. We also evaluate the accuracy loss caused by GA.

A. *Overall Performance Results*

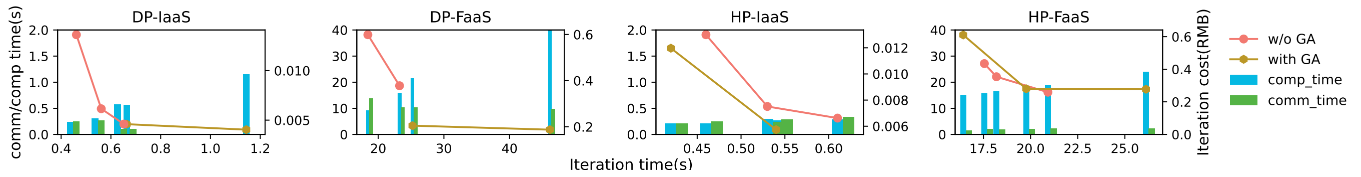
In different distributed training scenarios, we carried out cost and time optimization for the training processes with and without GA. These scenarios include two different distributed parallelism strategies and two popular cloud computing services for training four different models. The overall performance results are shown in Figure 4. We choose the commonly used *gbs* 256 for the experiment, then the cost-time performance of whether using GA is tested under two different cloud computing services of IaaS and FaaS, and two different parallel strategies of DP and HP. We use yellow and red polylines to represent the optimal performance achieved with and without GA under different weights. For each set of optimized resource configurations, we also measure the computation time and communication time during its training process, represented by blue and green bars in the figure, respectively. Note that although four sets of weight values are set before optimization, we often observe only 2 or 3 points in different Pareto frontier curves, as some weight value settings may lead to worse performance in the optimization process.

From the experimental results, it can be observed that the incorporation of GA can improve the cost-time optimization in most scenarios: in the DP scenario, the usage of GA tends to bring lower cost options for the training process, while in the HP scenario, GA usually makes the training cost and training time further optimized at the same time. Nevertheless, there are still some training scenarios that do not follow the aforementioned trends (e.g., the difference in the optimal performance obtained with or without GA when training *Resnet101* with HP strategy under FaaS is not significant). We will discuss and analyze the reasons for this result in DP and HP scenarios separately and explain the suitable scenarios for using GA.

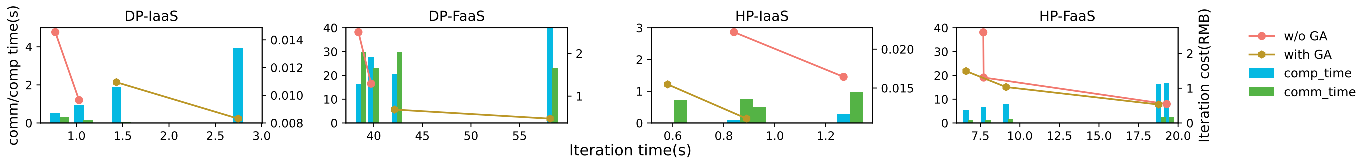
The appropriate scenarios under DP strategy. In our experiments, we can observe that under the DP scenario, the incorporation of GA can lead to a further decrease in the optimal training cost, albeit at the cost of increased total training time. Specifically, GA can result in a reduction in cost ranging from 3.8% to 21.5% for IaaS, but the training time will increase by up to 75.4% compared to training without GA. In contrast, GA has a more significant effect on reducing training cost in FaaS, resulting in an average reduction of 31.2% for 4 different workloads, with only a corresponding 17.3% average increase in training time.

As mentioned in §III, with the increasing number of DP nodes, the communication time during each iteration's training will also increase, and eventually affect the total training cost, making the relationship between minimal iteration cost and *gbs* show a superlinear growth. However, GA brings us the possibility to make tradeoffs between the number of workers and the communication time, making it possible to further optimize the training cost, albeit with a slight increase in the training time. It can be easily observed from Figure 4 that the optimization algorithm is able to find a configuration solution with lower cost under GA, which corroborates our above discussion. We can also draw this conclusion from the relative length of the computation time and communication time: the yellow curve represented by using GA tends to have higher computation time and lower communication time compared to the red curve in the above results plot.

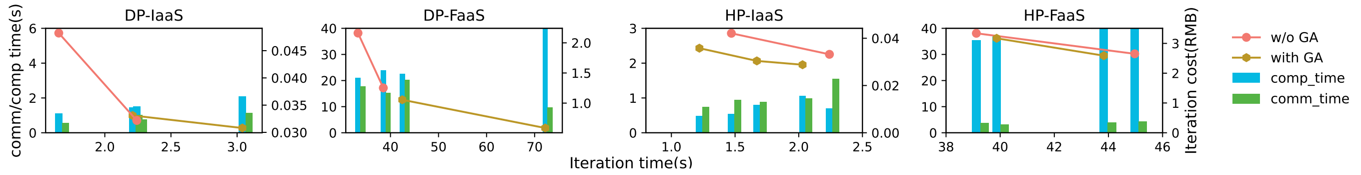
In addition, it is worth noting that in the above experiments we used a relatively large *gbs* of 256 so that GA is able to play a role in obtaining lower training cost for different DP scenarios. If the *gbs* is small enough, some small models can be trained on a single node without distributed processing. At this time, GA will not produce the corresponding optimization effect. For example, we show the case of *gbs*=64 training Resnet101 using IaaS in Figure 5. Since a single A10 GPU



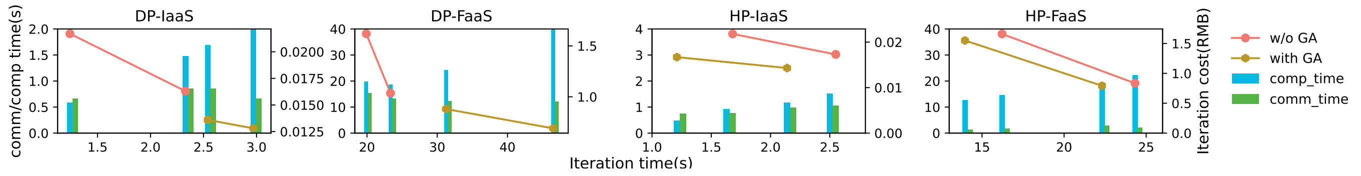
(a) Resnet101



(b) Amoebanet-D18



(c) Amoebanet-D36



(d) BERT-Large

Fig. 4: **The overall performance comparison whether GA is used or not.** Global batch size = 256 is set in this experiment.

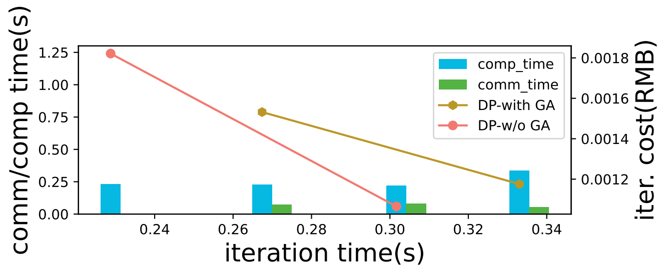
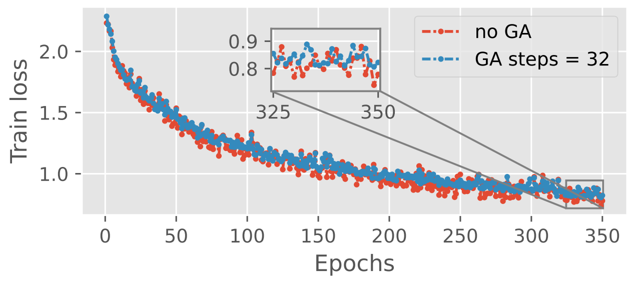


Fig. 5: **Optimization results for *Resnet101* with DP, with *gbs*=64.** Under this scenario, applying GA can not even reach the vanilla optimal performance without GA.

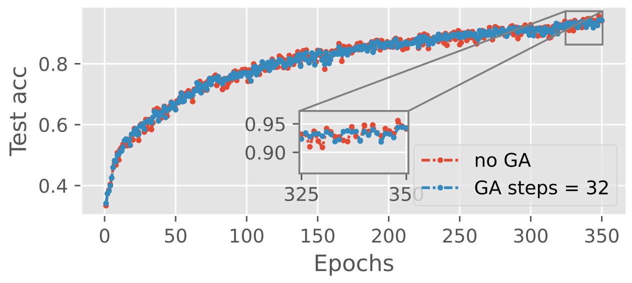
node is sufficient to support the workload with *gbs*=64, we could not directly apply GA to this configuration scheme. Although GA allows us to select some smaller instances for training, better training performance was not observed in the actual experimental tests.

The appropriate scenarios under HP strategy. In the HP-IaaS scenario, we can observe an average 21.2% decrease in the optimal training time after considering *mbs* as an optimization variable, accompanied by an average 24.8% decrease in the training cost when training different models. However, when using HP-FaaS, the optimization effect is not significant compared to IaaS situation, especially when training smaller models like *Resnet101* and *Amoebanet-D18*.

Our experiment further shows that the difference between the optimal *mbs* and the default *mbs* of the popular pipeline training framework is more pronounced in the IaaS scenario compared to the FaaS scenario. This distinction can be explained not only by the alteration of the training framework discussed in §III, but also by the elevated bandwidth and GPU acceleration capabilities present in IaaS scenarios, as demonstrated by the monotonicity of equations (2) and (3). For FaaS, the optimal *mbs* setting is similar to the default setting, which leads to a poor optimization effect when utilizing GA. Note that when training large models, the coefficient of the part that cannot be accelerated by parallelization in the time expression will also increase, which will shift the position of the optimal *mbs* to the right. Therefore, in the above experiment, we can observe that GA has better optimization effects during the training of large models.



(a) Training loss



(b) Test accuracy

Fig. 6: **Convergence rate and accuracy results for *Resnet101*, using *gbs* = 256 and *Cifar-10* dataset.**

TABLE V: **Final training accuracy loss caused by GA.**

GA steps	Final accuracy (%) for Resnet101	Final F1-score(%) for BERT-Large
1	94.42	90.91
2	94.42	90.72
4	94.39	90.88
8	94.35	90.75
16	94.31	90.63
32	94.11	90.89

B. *Final Accuracy and Convergence Rate*

To determine the extent of the accuracy loss caused by GA, we tested the final accuracy and F1-score for *Resnet101* and *BERT-Large*, respectively, using a fixed *gbs* value of 256 and different number of GA steps. We follow the Linear Scaling Rule [39] while changing the *mbs*, which dictates that when the *mbs* is multiplied by a factor of K, the learning rate should be increased correspondingly by the same factor of K. The experiments are conducted on AliCloud gn7e.4xlarge instance and the results are shown in Table V[3]. Note that the accuracy of *Resnet101* decreases only slightly as we increase the GA steps, the loss being 0.31% with 32 GA steps. The F1-score of *BERT-Large* was not significantly impacted either. This finding illustrates that the cross-batch computation layers used in the model, such as the BN layer, may cause the trend of accuracy degradation in GA. Furthermore, GA does not cause significant accuracy degradation for models without these layers.

By comparing the convergence rate of training loss and test accuracy of *Resnet101* with and without GA in Figure 6, it can be found that the final convergence rate is not significantly impacted even with an increase in the number of GA steps to 32. While previous work [4] observes a higher accuracy loss with the increase of GA steps, we believe that the loss observed there is more likely to arise from the difference in synchronization schemes (synchronous vs asynchronous) rather than from GA. Moreover, it is worth noting that the Linear Scaling Rule plays a crucial role in the experiment. For example, fixing the learning rate while changing *mbs* may result in a maximum accuracy loss of up to 4%.

In conclusion, our experimental results illustrated that the direct use of GA does not significantly impact the final accuracy as well as the convergence rate. Although we observed a slight trend of accuracy degradation for models with cross-batch layers, there are rapidly developing normalization techniques, such as Instance Normalization [42] and Group Normalization [43], which are designed for stable performances independent of batch size. By replacing the normalization layers with these above layers during training, we can mitigate the decreasing accuracy trend in GA.

[3]Our final accuracy results are comparable to two popular baseline results: Pytorch-Cifar10 at 93.75% accuracy [40] and SQuAD-BERT at 90.96% F1-score [41].

VI. CONCLUSION

In this paper, we investigated the benefits of GA for cost-time optimization in four cloud-based distributed training scenarios with four representative DL frameworks. Through experiments and analysis, we showed that under the DP strategy, GA tends to have a significant optimization effect in scenarios with larger models and smaller bandwidths. Our findings demonstrated that the combination of GA and DP can reduce the training cost by 31.2% while increasing the total training time by 17.3% in some training scenarios. Under the HP strategy, considering *mbs* as an optimization variable is more suitable for fast computing/communicating and larger model training scenarios. For IaaS-based training, this reduced training time by 21.2% and cost by 24.8% on average. After evaluating and analyzing the performance of GA in various scenarios, we conclude that GA is a worthwhile consideration when designing a training framework for cost-time optimization. We also investigated the appropriate use cases for GA under different levels of parallelism, which may provide guidance for future design of training frameworks.

ACKNOWLEDGMENT

We thank the anonymous reviewers for their constructive reviews. This work is partly supported by the National Natural Science Foundation of China (No. 62072302, 61960206002), the US National Science Foundation under Grants 2105564, 2236987, and VMWare.

REFERENCES

[1] Pytorch, "Gradient accumulation pytorch," https://gist.github.com/thomwolf/ac7a7da6b1888c2eeac8ac8b9b05d3d3.

[2] Tensorflow, "Gradient accumulation tensorflow," https://github.com/tensorflow/tensorflow/pull/32576.

[3] T. D. Le, T. Sekiyama, Y. Negishi, H. Imai, and K. Kawachiya, "Involving cpus into multi-gpu deep learning," in *Proceedings of the 2018 ACM/SPEC international conference on performance engineering*, 2018, pp. 56–67.

[4] I. Hakimi, R. Z. Aviv, K. Y. Levy, and A. Schuster, "Laga: Lagged allreduce with gradient accumulation for minimal idle time," in *2021 IEEE International Conference on Data Mining (ICDM)*. IEEE, 2021, pp. 171–180.

[5] S. Li, Y. Zhao, R. Varma, O. Salpekar, P. Noordhuis, T. Li, A. Paszke, J. Smith, B. Vaughan, P. Damania *et al.*, "Pytorch distributed: Experiences on accelerating data parallel training," *arXiv preprint arXiv:2006.15704*, 2020.

[6] Alibaba, "Alibaba cloud elastic compute service," https://www.aliyun.com/product/ecs.

[7] Amazon, "Amazon elastic compute cloud(ec2)," https://aws.amazon.com/aws/ec2.

[8] Microsoft, "Microsoft azure cloud vm," https://azure/microsoft.com/services/vm.

[9] S. Fan, Y. Rong, C. Meng, Z. Cao, S. Wang, Z. Zheng, C. Wu, G. Long, J. Yang, L. Xia *et al.*, "Dapple: A pipelined data parallel approach for training large models," in *Proceedings of the 26th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, 2021, pp. 431–445.

[10] S. Li, R. J. Walls, L. Xu, and T. Guo, "Speeding up Deep Learning with Transient Servers," in *2019 IEEE International Conference on Autonomic Computing (ICAC)*, Jun. 2019, pp. 125–135.

[11] Alibaba, "Alibaba cloud function compute," https://www.aliyun.com/product/fc.

[12] Amazon, "Aws lambda," https://aws.amazon.com/lambda/.

[13] Microsoft, "Microsoft azure cloud computing," https://azure.microsoft.com/.

[14] Y. Liu, B. Jiang, T. Guo, Z. Huang, W. Ma, X. Wang, and C. Zhou, "Funcpipe: A pipelined serverless framework for fast and cost-efficient training of deep learning models," *arXiv preprint arXiv:2204.13561*, 2022.

[15] J. Jiang, S. Gan, Y. Liu, F. Wang, G. Alonso, A. Klimovic, A. Singla, W. Wu, and C. Zhang, "Towards demystifying serverless machine learning training," in *Proceedings of the 2021 International Conference on Management of Data*, 2021, pp. 857–871.

[16] T. Ben-Nun and T. Hoefler, "Demystifying parallel and distributed deep learning: An in-depth concurrency analysis," *ACM Computing Surveys (CSUR)*, vol. 52, no. 4, pp. 1–43, 2019.

[17] Y. Huang, Y. Cheng, A. Bapna, O. Firat, D. Chen, M. Chen, H. Lee, J. Ngiam, Q. V. Le, Y. Wu *et al.*, "Gpipe: Efficient training of giant neural networks using pipeline parallelism," *Advances in neural information processing systems*, vol. 32, 2019.

[18] A. Ali, S. Zawad, P. Aditya, I. E. Akkus, R. Chen, and F. Yan, "Smlt: A serverless framework for scalable and adaptive machine learning design and training," *arXiv preprint arXiv:2205.01853*, 2022.

[19] D. Rani and R. K. Ranjan, "A comparative study of saas, paas and iaas in cloud computing," *International Journal of Advanced Research in Computer Science and Software Engineering*, vol. 4, no. 6, 2014.

[20] O. Alipourfard, H. H. Liu, J. Chen, S. Venkataraman, M. Yu, and M. Zhang, "Cherrypick: Adaptively unearthing the best cloud configurations for big data analytics," in *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17)*, 2017, pp. 469–482.

[21] J. Yi, C. Zhang, W. Wang, C. Li, and F. Yan, "Not all explorations are equal: Harnessing heterogeneous profiling cost for efficient mlaas training," in *2020 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. IEEE, 2020, pp. 419–428.

[22] C. J. Shallue, J. Lee, J. Antognini, J. Sohl-Dickstein, R. Frostig, and G. E. Dahl, "Measuring the effects of data parallelism on neural network training," *arXiv preprint arXiv:1811.03600*, 2018.

[23] T. Chen, B. Xu, C. Zhang, and C. Guestrin, "Training deep nets with sublinear memory cost," *arXiv preprint arXiv:1604.06174*, 2016.

[24] Z. Bian, Q. Xu, B. Wang, and Y. You, "Maximizing parallelism in distributed training for huge neural networks," *arXiv preprint arXiv:2105.14450*, 2021.

[25] C.-C. Chen, C.-L. Yang, and H.-Y. Cheng, "Efficient and robust parallel dnn training through model parallelism on multi-gpu platform," *arXiv preprint arXiv:1809.02839*, 2018.

[26] M. Shoeybi, M. Patwary, R. Puri, P. LeGresley, J. Casper, and B. Catanzaro, "Megatron-lm: Training multi-billion parameter language models using model parallelism," *arXiv preprint arXiv:1909.08053*, 2019.

[27] N. Shazeer, Y. Cheng, N. Parmar, D. Tran, A. Vaswani, P. Koanantakool, P. Hawkins, H. Lee, M. Hong, C. Young *et al.*, "Mesh-tensorflow: Deep learning for supercomputers," *Advances in neural information processing systems*, vol. 31, 2018.

[28] D. Narayanan, A. Harlap, A. Phanishayee, V. Seshadri, N. R. Devanur, G. R. Ganger, P. B. Gibbons, and M. Zaharia, "Pipedream: generalized pipeline parallelism for dnn training," in *Proceedings of the 27th ACM Symposium on Operating Systems Principles*, 2019, pp. 1–15.

[29] Z. Luo, X. Yi, G. Long, S. Fan, C. Wu, J. Yang, and W. Lin, "Efficient pipeline planning for expedited distributed dnn training," *arXiv preprint arXiv:2204.10562*, 2022.

[30] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in *International conference on machine learning*. PMLR, 2015, pp. 448–456.

[31] J. Lamy-poirier, "Layered gradient accumulation and modular pipeline parallelism for improved training of machine learning models," Dec. 1 2022, uS Patent App. 17/668,200.

[32] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[33] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," *arXiv preprint arXiv:1810.04805*, 2018.

[34] E. Real, A. Aggarwal, Y. Huang, and Q. V. Le, "Regularized evolution for image classifier architecture search," in *Proceedings of the aaai conference on artificial intelligence*, vol. 33, no. 01, 2019, pp. 4780–4789.

[35] Stanford, "The stanford question answering dataset," https://rajpurkar.github.io/SQuAD-explorer/.

[36] Alibaba, "Alibaba cloud object storage service," https://www.aliyun.com/product/oss.

[37] J. Snoek, H. Larochelle, and R. P. Adams, "Practical bayesian optimization of machine learning algorithms," *Advances in neural information processing systems*, vol. 25, 2012.

[38] E. Brochu, V. M. Cora, and N. De Freitas, "A tutorial on bayesian optimization of expensive cost functions, with application to active user modeling and hierarchical reinforcement learning," *arXiv preprint arXiv:1012.2599*, 2010.

[39] 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.

[40] kuangliu, "Train cifar10 with pytorch," https://github.com/kuangliu/pytorch-cifar.

[41] NVIDIA, "Nvidia deep learning examples for tensor cores," https://github.com/NVIDIA/DeepLearningExamples/tree/master/PyTorch/LanguageModeling/BERT.

[42] D. Ulyanov, A. Vedaldi, and V. Lempitsky, "Instance normalization: The missing ingredient for fast stylization," *arXiv preprint arXiv:1607.08022*, 2016.

[43] Y. Wu and K. He, "Group normalization," in *Proceedings of the European conference on computer vision (ECCV)*, 2018, pp. 3–19.