Empirical Analysis and Modeling of Compute Times of CNN Operations on AWS Cloud

Ubaid Ullah Hafeez, Anshul Gandhi
PACE Lab, Computer Science Department
Stony Brook University, Stony Brook, NY, USA
uhafeez@cs.stonybrook.edu, anshul@cs.stonybrook.edu

***Abstract*—Given the widespread use of Convolutional Neural Networks (CNNs) in image classification applications, cloud providers now routinely offer several GPU-equipped instances with varying price points and hardware specifications. From a practitioner's perspective, given an arbitrary CNN, it is not obvious which GPU instance should be employed to minimize the model training time and/or rental cost. This paper presents Ceer, a model-driven approach to determine the optimal GPU instance(s) for any given CNN. Based on an operation-level empirical analysis of various CNNs, we develop regression models for heavy GPU operations (where input size is a key feature) and employ the sample median estimator for light GPU and CPU operations. To estimate the communication overhead between CPU and GPU(s), especially in the case of multi-GPU training, we develop a model that relates this communication overhead to the number of model parameters in the CNN. Evaluation results on AWS Cloud show that Ceer can accurately predict training time and cost (less than 5% average prediction error) across CNNs, enabling 36%–44% cost savings over simpler strategies that employ the cheapest or the latest generation GPU instances.**

I. Introduction

The past few years have seen a significant increase in the adoption of Deep Neural Networks (DNNs) for a wide range of machine learning applications [1], [2], [3]. The success of DNNs is primarily due to their large models which have the capacity to learn complex features from big data. Convolutional Neural Networks (CNNs) are a popular and efficient class of DNNs that employ the convolution operation for processing data that has a known grid-like topology, such as images, making CNNs among the most popular techniques for image classification [4], [5].

Despite their success, CNNs, and DNNs in general, are expensive models for training owing to their large compute requirements and numerous parameters. Most CNN models are trained, using training frameworks such as TensorFlow [6], on machines equipped with expensive GPUs; see Section II for more details on CNN training. Modern servers equipped with commodity GPUs can be prohibitively expensive; for example, a 4-GPU (NVIDIA Tesla V100) NVLink-equipped server can easily cost upwards of $35,000 [7].

A promising alternative to purchasing such expensive servers is to rent similar cloud resources for the duration of model training. Most public cloud providers, including AWS, Google Cloud, and Azure, now provide several GPU-equipped Virtual Machine (VM) offerings [8], [9], [10]. Specifically, AWS currently offers VM instances that support four different GPU models [8] with hourly rental costs of a basic single-GPU VM instance ranging from $0.75 to $3.06 [11].

Within the context of cloud-deployed CNN model training, an important concern is determining the best GPU instance (or VM) to employ, from among the available options, for a given CNN model to minimize the completion time and/or resource rental costs. For realistic model training, the training time (or completion time) can last from a few minutes to several days [12], [13], owing to the numerous model parameters that need to be learned; consequently, training a single model can be an expensive undertaking. As a specific example, the winner of the ImageNet challenge 2017, Squeeze-and-Excitation Networks (SENets), employed 145.8 million parameters with a training time of at least 250 GPU-hours [14]. Based on the rental cost of a similar AWS GPU model instance as employed by the above work [15], the model training cost can be estimated to be as high as $750.

Given an arbitrary CNN, deciding which GPU instance to employ to minimize the training time and/or rental cost is a non-trivial problem because of the following challenges:

- The available GPU instances may offer widely varying *price points* and *hardware specifications*. A more expensive instance may not necessarily result in the smallest training time, and vice-versa. Thus, simple strategies such as employing the cheapest instance or employing the latest GPU offering need not be optimal.
- CNNs can be functionally and structurally very *different* from each other, making it difficult to predict the training time of a given CNN based on performance analyses of other CNNs, even if on the same GPU model. Further, the training time of a CNN depends critically on its *input (or training) data size*.
- To reduce training time, practitioners often employ multiple GPUs and train the model in parallel over partitions of input data; this practice is referred to as *data parallelism* [16]. However, the training time under data parallelism need *not* scale perfectly with the number of GPUs due to communication overheads between CPU and GPUs. Worse, this overhead is GPU- and model-specific, making it difficult to predict the training time when using multiple GPUs.

Most of the prior work in this area has focused on the empirical analysis of DNN end-to-end training time on specific

GPU devices [5], [17]. By contrast, the cloud-deployed GPU instance cost optimization problem we consider in this paper requires an understanding of the constituent *operation-level* compute times (that make up the training time) across different commodity GPUs offered by cloud providers. Further, while much of the prior work has focused on speeding up the training time of a specific model under consideration [18], [19], we are more concerned with finding the optimal GPU instance for an *arbitrary* CNN. While some prior works aim to predict training time for arbitrary CNNs [4], [20], they only focus on layer-level modeling (a layer is a set of operations) and ignore small operations, which, as we discuss in Section IV, significantly hurts prediction accuracy.

In this paper, we perform a detailed empirical study of operation-level compute times and compute costs of several frequently used CNN models on all four publicly available GPU model types in the AWS cloud. Our analysis yields insights on the relationship between specific GPU hardware characteristics and the type of compute operations in the CNN model. For example, while the latest generation of GPU model instances (P3) are better suited, in terms of cost and performance, for memory-intensive operations (e.g., MaxPool-Grad), older generation of GPU instances (e.g., G4) are more cost-efficient for moderately compute-intensive operations. We also make the key finding that the set of *unique* operations that contribute to much of the training time across CNNs is typically small, and the compute time of these "heavy" operations exhibits *low variability*.

Based on our empirical analysis, we develop a model-driven approach, Ceer (a seer for CNNs), to accurately predict the training time and training cost of CNNs across GPU instances. The primary component of Ceer is the regression models for heavy operations that relate their compute time to input size for each GPU model; while linear regression works well for most operations, quadratic models are required for a few operations. Combined with our empirical finding that (present-day) CNNs are typically composed of the same set of these heavy operations, our regression models allow Ceer to recommend the optimal GPU instance choice for *any* CNN.

However, we find that only considering the heavy operations does not provide a good prediction accuracy for training time. While other "light" operations and operations that execute on the CPU often have small compute times, they do exhibit high variability, and so ignoring these operations hurts prediction accuracy by 15–25%. Instead of exhaustively modeling each of these operations for every GPU, Ceer uses a much simpler and practical approach, resulting in accurate training time and training cost predictions.

Finally, and importantly, to predict training time on multi-GPU instances, we develop a *CNN-oblivious* model that accurately estimates the communication delay between CPU and GPUs in each training iteration. Interestingly, the communication between CPU and GPU must also be taken into account for accurately predicting the training time on single-GPU instances; ignoring this overhead can increase prediction error by 20–30% for certain CNNs.

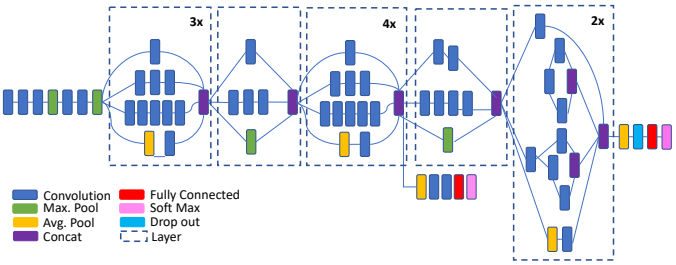

Fig. 1: Illustration of the DAG for the Inception-v3 CNN.

We evaluate Ceer on AWS cloud and show that Ceer consistently recommends the optimal GPU instance deployment in *every* scenario we consider and across (previously unseen) CNNs. This is enabled by the accurate training time and training cost models employed by Ceer; our average (test set) prediction error across CNNs and GPU instance types is about 4.2%. Compared to the strategy of picking the cheapest or most expensive (thus, latest) GPU model, Ceer can reduce rental costs by as much as 36% and 44%, respectively; alternatively, for a given cost budget, Ceer can reduce the corresponding training time by 89% and 48%.

The rest of this paper is organized as follows. Section II provides the necessary background on CNN training and describes the commercially available GPU model types on AWS cloud. Section III presents our empirical study of the operation-level compute times and compute costs of various CNN models on all AWS GPU model types. Section IV describes our modeling approach, Ceer, that accurately predicts the training time and cost for an arbitrary CNN. We evaluate the efficacy of Ceer in Section V. Finally, we discuss related work in Section VII and conclude the paper in Section VIII.

II. Background and Overview

This section provides the necessary background on CNNs and CNN training, and then discusses the various GPU models offered by AWS that we employ in our empirical study.

Convolutional Neural Networks (CNNs). A CNN is a convolutional neural network model with multiple layers that can learn the (possibly non-linear) relationship between the input features and the output variable, often expressed as model parameters or weights, based on training over (typically) labeled data sets. As the name indicates, CNNs employ a mathematical operation called convolution in place of general matrix multiplication in at least one of the layers; convolutions are more memory- and compute-efficient as compared to general matrix multiplication [21]. Further, almost all CNNs employ another mathematical operation called pooling as well; pooling operations enable CNNs to reduce the number of model parameters and computations, while avoiding overfitting [21]. The CNN model consists of sequence of layers of different types, with each layer consisting of several nodes, corresponding to computational operations. In this work, we specifically focus on CNNs given their popularity in practice.

CNN training. Modern CNN training frameworks, such as TensorFlow [6], model CNNs as directed acyclic graphs

(DAGs) where each graph node is a compute operation (e.g., multiplication, gradient calculation), typically run on a GPU or CPU, and each edge represents the data communication between operations. Figure 1 shows the underlying DAG of the Inception-v3 CNN model [22]. Here, the edges connecting different rectangles denote data flow between them. The colored rectangles in the figure represent a functional compute unit (see legend), which we refer to in this paper as an ***operation***. Note the $\times$ multiplier in some of the (dotted boundary) layers; these indicate that the layer repeats several times in sequence.

Training generally consists of multiple iterations of the training data over the same model. For example, if the total training data set size is D samples, and each iteration processes a batch size of B samples, then the training will complete in D/B iterations. Some data is typically transferred between the CPU and GPU at the start and end of each iteration. The entire training may be repeated multiple times in epochs until a convergence criteria is reached [13].

A commonly employed technique to enhance CNN training across GPUs is *data parallelism*. Under data parallelism, multiple GPUs are employed, each with a complete replica of the CNN model, to process subsets of the input training data in parallel. The learned weight updates from each GPU are periodically aggregated in a synchronization phase and updated for subsequent iterations; the communication overhead for synchronization can be substantial [23].

AWS GPU models available for CNN training. Several commercial cloud service providers, including AWS [8], Google Cloud [9], and Azure [10], now offer GPU instances (as VMs) to customers. In this work, we focus specifically on AWS, the largest commercial cloud resource provider [24], [25]. AWS offers various GPU instance types, each consisting of one of the following GPU models [8]:

- *NVIDIA Tesla V100* (P3 instances) with 5,120 CUDA Cores, 640 Tensor Cores, and (default of) 16GB of GPU memory.
- *NVIDIA K80* (P2 instances) with 2,496 parallel processing cores and 12GB of GPU memory.
- *NVIDIA T4 Tensor Core* (G4 instances) with 2,560 parallel processing cores and (default of) 16 GB of memory.
- *NVIDIA Tesla M60* (G3 instances) with 2,048 parallel processing cores and 8 GB of GPU memory.

While each GPU model is offered in different instance sizes, the corresponding basic 1-GPU models we consider for the above four GPU models, along with their hourly (On-Demand Pricing) rental costs [11], are : p3.2xlarge ($3.06/hr), p2.xlarge ($0.90/hr), g4dn.2xlarge ($0.752/hr), and g3s.xlarge ($0.75/hr). We also employ multi-GPU versions of these instances in our experimental evaluation (see Section V). We note that AWS also offers instances with custom-built chips or FPGAs, but we omit these instances in our discussion and evaluation given our focus on GPUs.

III. Empirical Analysis

This section presents one of our main contributions — an empirical study of compute times of CNN operations on all four GPU model types offered by AWS. We first present our results for operation-level compute times and compute costs, and then discuss how these are impacted by the input size. We then analyze how training time scales with the number of GPUs (under data parallelism).

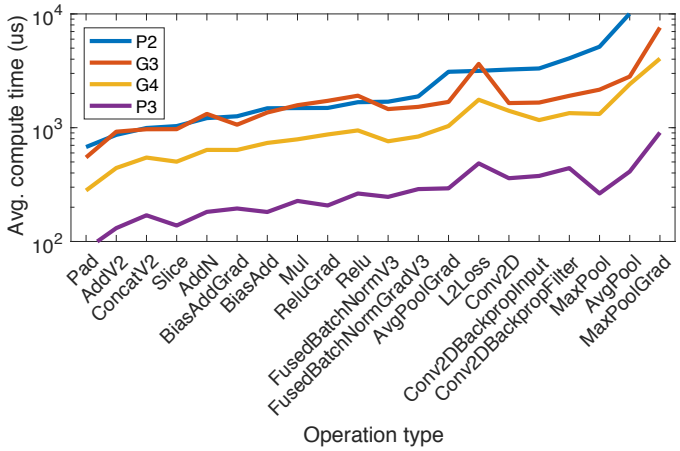

Fig. 2: Average compute times of different "heavy" operations for 4 different AWS GPU model types.

For our empirical study, we consider the following popularly employed CNN models, which together provide us with 12 distinct CNNs. We obtain empirical results by training these CNNs on TensorFlow [6] on AWS GPU instances.

- *VGG* is a CNN with multiple convolutional, pooling, and fully connected layers, and is often used for image classification [23]. We experiment with three different variants having varying numbers of convolutional layers, i.e., VGG-11 (with 11 layers), VGG-16, and VGG-19.
- *Inception* is a more memory-efficient version of VGG. We experiment with three versions of Inception – v1, v3, and v4, each with a different number and size of convolutions.
- *ResNet* can provide better accuracy in image classification as it employs residual networks and shortcut connections. We experiment with four popular variants of ResNet-v2 – 50-layered, 101-layered, 152-layered and 200-layered.
- *Inception-ResNet-v2* is similar to Inception-v3, but augmented with shortcut connections.
- *AlexNet* mainly consists of convolutions and fully connected layers, and is one of the very first DNNs developed for image classification.

From these 12 CNN models, without loss of generality, we select Inception-v3, AlexNet, ResNet-101, and VGG-19 as our test set, and the remaining 8 CNNs as our training set. All empirical results in this section employ the 8 training set CNNs; we employ the test set CNNs for model validation and evaluation in Sections IV and V, respectively.

A. Operation-level compute times

CNN models are composed of numerous operations; however, the number of *unique* operation types across CNNs is typically small. For example, while the simplified Inception-v3 model, shown in Figure 1, has numerous operations (denoted as

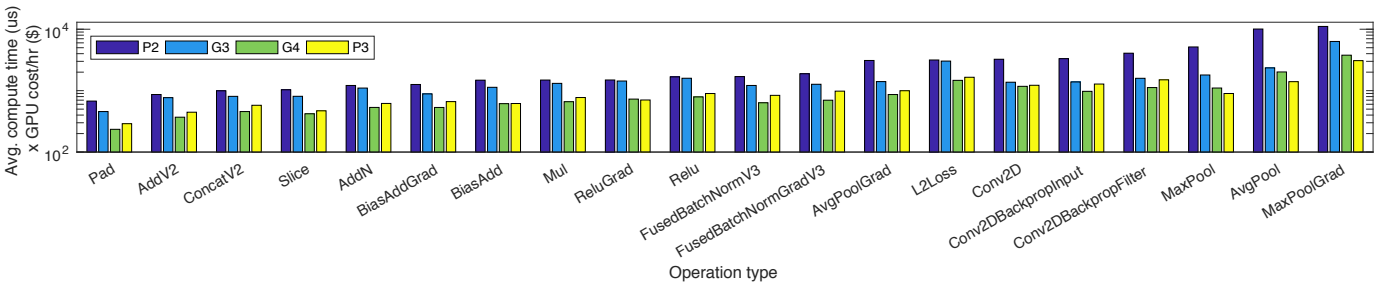

Fig. 3: Cost of operations (product of compute time and rental cost of GPU instance), on log scale, for different GPU models.

rectangles), the number of unique operations, represented by their color, is fairly small. Figure 2 shows the compute time, on log scale, for different GPU operations types on all four AWS GPU model type instances discussed in Section II. The compute times shown in the figure are averaged over 1,000 iterations of each of the 8 training set CNNs identified above. For ease of presentation, in the rest of this section, we omit operations that have negligible compute times (< 0.5 ms on P2), unless mentioned otherwise. We refer to these operations that have small compute times as ***light*** operations; we refer to the other GPU operations, shown in Figure 2, as ***heavy*** operations. Together, the light operations contribute to less than 7% of the model training time.

We see that there is an almost consistent relative ranking of compute times across GPU models, with P3 having the lowest compute times and P2 almost always having the highest compute times; for some operations, G3 has higher compute times than P2. Averaging across all (heavy) operations, P3 provides an impressive $10\times$ lower compute time compared to P2, and an almost $4\times$ lower compute time compared to G4. While the compute times for P2 and G3 appear to be close in the figure (note the log scale), P2 results in almost 50% higher compute time, on average, compared to G3.

In terms of operations, the pooling operations have high compute times, whereas the simpler Add-type operations have lower compute times. This is because, unlike the simple Add operations, the pooling operations involve a combination of operations such as concatenation, addition, and multiplication, and also involve more reads and writes to GPU memory. We also find that the compute time of operations often scales with their input size. For example, in Figure 2, the AddV2 operation has an average input size of about 85MB whereas the BiasAdd operation has an average input size of about 120MB. While the functionality of these two operations is very similar, BiasAdd has higher compute times because of the larger input size (since one of the inputs is typically a matrix).

Focusing on operation-level compute times has the advantage that by understanding these constituent compute times, we can estimate the training time per iteration for *any* arbitrary CNN, given that CNNs are typically composed of the same small set of unique operations. For example, the 20 heavy operations shown in Figure 2 contribute to 47%–94% of the training time of our training set CNNs. Of course, for some CNNs, only focusing on heavy operations will not suffice.

B. Operation-level compute costs

In addition to compute time, the *cost of the GPU instance* is also important when determining the choice of GPU instance to employ. Figure 3 shows the rental cost (normalized by the number of μs in an hour, 3.6×10^9) incurred when running the GPU operation on a specific GPU model type (assuming the basic, single-GPU instance cost), over the duration of its compute time. The values shown in this figure are obtained by multiplying the average compute times from Figure 2 with the corresponding GPU instance price.

In terms of cost over the duration of compute time, we see that P3 and G4 are typically the best choices, depending on the operation. However, compared to the relative improvements in Figure 2, the cost benefit afforded by P3 is not as pronounced. In fact, while P3 lowers the compute time by about $4\times$, on average, compared to G4, the average cost (averaged over all operations) over the compute time duration is slightly lower for G4 compared to P3. Likewise, the $10\times$ compute time improvement afforded by P3 over P2 translates to a much lower $3\times$ when considering the average cost.

For the 20 operations shown in Figure 3, G4 provides the lowest cost for 16 of these operations while P3 provides the lowest cost for the remaining 4 operations. In particular, for the pooling operations, we find that P3 lowers the cost by about 20%, on average, compared to G4; the peak reduction, for AvgPool, is 31%. For the 16 operations where G4 provides a lower cost, the average cost reduction over P3 is about 16%; the peak reduction, for FusedBatchNormGradV3, is about 29%. The GPU model supported by P3 instances (NVIDIA Tesla V100) has high compute power and memory bandwidth, and is thus well suited for the memory-intensive pooling operations. While G4 is less powerful than P3, it has a much lower cost than P3 instances, and so is more cost-efficient for operations that are not too compute intensive.

C. Impact of input size on compute times

Thus far, we have considered average compute times for each operation. However, each operation in a CNN operates over a certain input data, whose *data size* can dictate the execution time of the operation. The dots in Figure 4 show the compute time of the ReLU operation (that implements the rectified linear activation unit activation function) for different GPU model types. Clearly, the compute time depends on the input data size. The solid lines refer to our modeling results, and will be discussed in the next section.

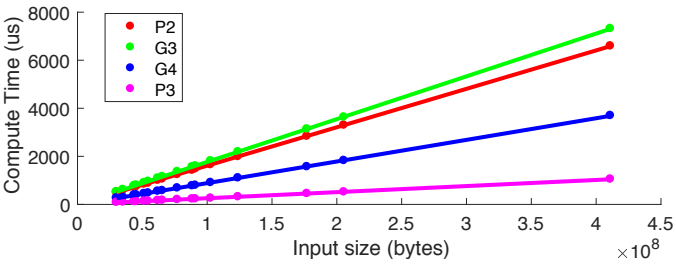

Fig. 4: Compute time of ReLU operation as a function of input size for different GPU models. Also shown, with lines, is the linear regression fit based on the empirical observations (dots).

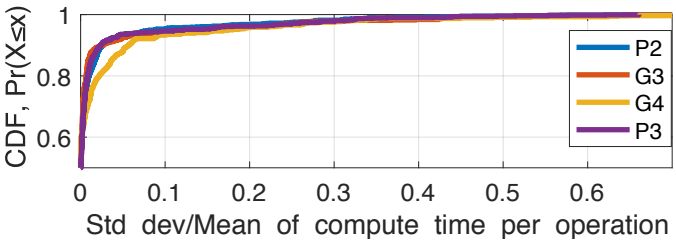

Fig. 5: CDF of normalized standard deviation of (heavy) operations for different GPU models.

We also repeated the above input size analysis for other heavy operations and found that, in all cases, the compute time depends significantly on the input sizes. For some operations (e.g., Conv2D, AvgPool, etc.), the compute time also depends on the size of supplemental inputs, such as filters, strides, and padding, in addition to the size of the input images.

While the compute times depend on the input data size, fortunately, for each unique heavy GPU operation and unique input data size, the compute time is largely invariant across runs. Figure 5 shows the CDF of normalized standard deviation (normalized by the mean) of compute times of different heavy GPU operations with unique input data size over their 1,000 iterations, for each GPU model type. We see that the normalized deviation is typically quite low, with 95% of the values being less than 0.1. Note that we are omitting light operations (those that have negligible compute times) and CPU operations; we find that these operations exhibit higher normalized deviation than heavy GPU operations. We discuss the impact of this higher standard deviation of light GPU and CPU operations on the design of Ceer in Section IV-D.

D. Scaling of model training time with data parallelism

Thus far, we have only considered the basic AWS GPU instance, with a single GPU, for each GPU model type. In this subsection, we analyze how model training time scales under data parallelism as we employ multiple GPUs. For the empirical analysis in this subsection, we use the following AWS GPU instances, each of which has (at least) 4 GPUs of the same kind: p3.8xlarge ($12.24/hr), p2.8xlarge ($7.20/hr), g4dn.12xlarge ($3.912/hr), and g3.16xlarge ($4.56/hr).

Figure 6 shows the training time as a function of number of GPUs, under data parallelism, for different GPU model types using the Inception-v1 CNN. Here, we use an input data with 6,400 samples of ImageNet [26] for training. As expected, the training time drops as more GPUs are employed. Compared to the training time for 1 GPU, the training time under 2, 3, and 4 GPUs drops by around 35.8%, 46.6%, and 53.6%, respectively, averaged across all GPU types. While the reduction is significant when going from 1 GPU to 2 GPUs (35.8%), the reduction is less pronounced when going from 2 GPUs to 3 GPUs (16.9%) and from 3 GPUs to 4 GPUs (13.1%). The trend is qualitatively similar for other CNNs.

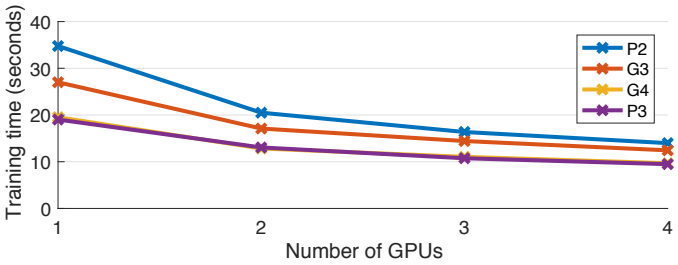

Fig. 6: Impact of number of GPUs, under data parallelism, on the training time for a sample input for different GPU model types for the Inception-v1 CNN.

The above results highlight the fact that adding more GPUs does help reduce training time, but this reduction is less pronounced for higher number of GPUs, suggesting diminishing returns. This is because of the synchronization phase [16], wherein each GPU communicates its parameter updates, and the next training iteration can only begin once all parameter updates have been received; as the number of GPUs increases, so does the probability of "stragglers". We note that other forms of parallelism can also be employed for model training across GPUs, such as model parallelism [27], [28], [29] and pipeline parallelism [30], [31] (see Section VII). However, data parallelism continues to be the default parallelization approach employed when training models across GPUs in frameworks such as TensorFlow [6].

Under data parallelism, the entire CNN model is replicated on each GPU [16]. Thus, the compute times of individual operations of the CNN follow the same trends on each individual GPU as observed for the single GPU instance in Section III-C (but with a possibly different input data size since the data is partitioned across the multiple GPUs under data parallelism, thereby reducing the batch size per GPU). Further, since each GPU deals exclusively with its data partition, summing up the constituent operations' compute times can provide a lower bound on the per-iteration training time, assuming a uniform partitioning of data over the multiple GPUs. This is only a lower bound since there is additional communication overhead in data parallelism, as we observed in Figure 6.

IV. Solution Design for Ceer

The key question we wish to address in this paper is "Given a CNN model, which GPU instance(s) should be employed to optimize the model training time and/or cost?" Towards providing a solution for this problem, our empirical study provides the following crucial insights, which together guide

the design of our model- and data-driven solution, Ceer:

(1) CNN models are composed of several operations, with a few unique "heavy" operation types contributing to the majority of the model training time.

(2) Different GPU models provide different cost and performance benefits for different operation types, and this tradeoff can be inferred from empirical data.

(3) For a given {heavy operation, input data size} pair, the compute times on a given GPU model have low variability, and can thus be accurately estimated based on the input size(s). By contrast, light operations have high variability in compute times.

(4) For a single GPU, the training time per iteration of the CNN can be estimated by summing up the compute times of constituent operations.

(5) Based on the observed overhead of data parallelism, the training time on multi-GPU instances can be estimated by extrapolating from the single-GPU estimate.

A. Training time and cost estimation with Ceer

For the case of a single GPU instance, based on insight (4), we can estimate the per-iteration training time for a given CNN as $\sum_{i=1}^{n} t_{GPU,op_i}(input_i)$, where i indexes over all n operations in the CNN and $t_{GPU,op_i}(input_i)$ is the function that estimates the compute time of the operation op_i (based on its operation type) with input size $input_i$ on GPU type GPU. Note that n includes *all* operations in the CNN, including heavy, light, and CPU operations; we discuss the $t_{GPU,op_i}()$ function modeling for these operations in Section IV-B.

To obtain the model training time from the per-iteration training time, we need to determine the number of iterations needed to process the total training data. If the total data is D units (samples or bytes), and the batch size per iteration (often a fixed value for a CNN) is B units, then the total number of iterations for training is D/B. Thus, the model training time (one epoch), T, for a given CNN executed on a GPU model type, GPU, can be estimated as:

$$T_{CNN,GPU} = \left(\sum_{i=1}^{n} t_{GPU,op_i}(input_i) \right) \cdot \frac{D}{B}, \qquad (1)$$

Unfortunately, the above training time estimate does not include the *communication time* between CPU and GPU, which we find is typically incurred at least at the start and end of each iteration. Ignoring this communication time, even for a single-GPU instance, can hurt training time prediction accuracy by 5–20%; for AlexNet, the prediction error when using Eq. (1), and thus ignoring the communication time, is almost 30%. Interestingly, for the case with $k > 1$ GPUs (under data parallelism), we also have to take into account the slowdown incurred in each iteration due to the communication overhead between GPUs. We thus denote the collective communication overhead (including between CPU and GPU, and between GPUs), which can be GPU- and CNN-dependent, by $S^k_{GPU}(CNN)$; we discuss the S^k_{GPU} function modeling in Section IV-C. The corrected training time, including for multi-GPU instances (under data parallelism), can then be estimated based on insight (5) as:

$$T^k_{CNN,GPU} = \left(S^k_{GPU}(CNN) + \sum_{i=1}^{n} t_{GPU,op_i}(input_i) \right) \frac{D}{k \cdot B} \qquad (2)$$

Note that the overhead is added for each iteration. Also note that we consider here that the batch size is the same for each GPU in the single- and multi-GPU settings, and so the number of iterations needed for training a given input data reduces by a factor k (since k times more data is being processed in each iteration across k GPUs). If batch size for each GPU is different between the single-GPU and multi-GPU setting, then the number of iterations can be modified accordingly.

Finally, the training cost can be estimated as $C^k_{CNN,GPU} = T^k_{CNN,GPU} \times c_{GPU,k}$, where $c_{GPU,k}$ is the cost per unit time of renting the k-GPU cloud instance (with $k \geq 1$) supporting GPU type GPU. These costs are usually published and easily available via the cloud provider [11].

B. Modeling the compute time

To determine $t_{GPU,op}(input)$, the compute time estimate of operation op on GPU type GPU with $input$ as the input size, we separately model heavy and light operations, given their distinct characteristics. Note that $input$ can be a vector; for example, for the Conv2D (2-D convolution, in TensorFlow) operation, the size of both input images and the size of the filters serve as $input$ to the compute time model, $t_{GPU,op}()$.

Heavy operations. We estimate the compute time of heavy operations by employing a regression model over the training data analyzed in Section III-A. Since different operations have different inputs, we build a different model for each heavy operation. We find that linear regression works well for most heavy operations given their observed linear relationship between compute time and input sizes; for example, see the linear regression fit denoted by solid lines in Figure 4. However, for a few operations, e.g., Conv2DBackpropFilter, a quadratic fit is much better suited; this is likely because these operations involve more complex mathematical logic beyond simple add and multiply operations [32].

The R^2 values for regression based on the training data (using the 8 training set CNNs) range from 0.84 to 0.98 across operations. When using the 4 test set CNNs for validation, the mean average prediction error across all heavy operation types ranges from 2% to 10%. One of the reasons for this high modeling accuracy is provided by insight (3), which states that the variability in compute times of a given {heavy operation type, input data size} pair is low.

Light operations and CPU operations. If we only include heavy operations in our training time model, Eq. (2), we find that the training time prediction error is moderately high (15–25%). While some of this error is to be expected since we are ignoring light operations (that contribute around 5–10% to training time), we find that there is another source of error

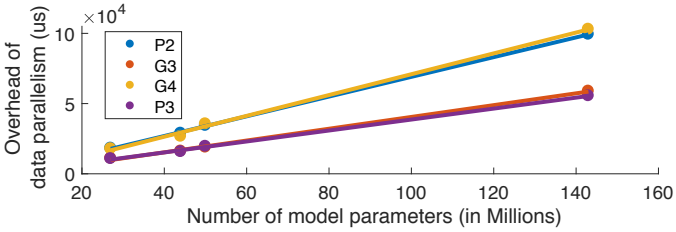

Fig. 7: Per-iteration communication overhead of data parallelism for 2 GPUs.

that we are ignoring — compute operations that execute on the CPU. When executing a CNN on a training framework, such as TensorFlow, some of the CNN DAG operations, e.g., SparseToDense, are executed on the CPU since they lack a GPU implementation.

The key challenge with extending the input-based regression model to light and CPU operations is that these operations have high variability, resulting in a poor regression fit. Since these operations do not significantly contribute to training time, we instead opt for a simpler approach to include their contribution in our training time estimate. Specifically, we use the *sample median* compute time of light operations, say $\tilde{t_l}$, and the median compute time of CPU operations, say $\tilde{t_c}$, computed over all instances of these operations in all training set CNNs across all GPU types, as an estimate of their compute time. Note that, for simplicity, these estimates are GPU-, CNN-, and operation-oblivious. We choose the median instead of the mean to avoid the unfair impact of possible outliers on the compute time estimate. Thus, we have:

$$t_{GPU,op}(input) = \tilde{t_l}, \quad \text{if } op \text{ is a light operation} \tag{3}$$

$$t_{GPU,op}(input) = \tilde{t_c}, \quad \text{if } op \text{ is a CPU operation} \tag{4}$$

In summary, if a CNN has n_h, n_l, and n_c heavy, light, and CPU operations, respectively, with $n = n_h + n_l + n_c$, then the light operations contribute $n_l \times \tilde{t_l}$ and the CPU operations contribute $n_c \times \tilde{t_c}$ to the per-iteration training time estimate. The contribution of each of the n_h heavy operations is accounted for by their individual linear regression models.

C. Modeling the communication overhead

Recall from Section III-D that there is a communication overhead penalty when employing data parallelism, which results in a sub-linear scaling of training time with the number of GPUs. The $S^k_{GPU}(CNN)$ function estimates this additional delay introduced in each iteration of the CNN model under the data parallelism approach when using $k > 1$ GPUs of type GPU. Since there is also some communication overhead between CPU and GPU for a single-GPU instance, we denote this overhead via the $S^k_{GPU}(CNN)$ function with $k = 1$. While it is possible to develop per-CNN models of communication overhead, we find that there is a CNN-oblivious modeling approach that works well in practice and is less restrictive to apply.

The markers (dots) in Figure 7 show the per-iteration communication overhead of data parallelism (including any synchronization delays) when employing 2 GPUs, as a function of the number of model parameters to be trained in the CNN; we observe a similar linear trend for 3 and 4 GPUs as well. Each value is empirically obtained for a given CNN by subtracting the average per-iteration training time for 1 GPU from the average per-iteration training time for multiple GPUs; the batch size per-GPU is kept the same in both configurations. If there was no communication overhead due to data parallelism, this difference would be zero. For the case of 1 GPU, we observe a similar trend when plotting the communication time (between CPU and GPU, obtained from GPU logs) and the number of model parameters.

Interestingly, when plotted against the number of model parameters, we find a nearly linear relationship for communication overhead, for every GPU model type. Given the observed linear behavior, we learn this relationship by employing simple linear regression over the training data. The R^2 values for regression for the different GPU models range from 0.88 to 0.98. The learned models for the per-iteration communication overhead for $k \geq 1$ GPUs of a given type refer to the $S^k_{GPU}()$ function, which takes as input the number of model parameters of the target CNN.

D. Optimal cloud instance recommendation via Ceer

At runtime, given an arbitrary CNN with training data size D and batch size (typically a default value) B, Ceer can provide the optimal recommendation for the cloud instance type that should be employed for training the given CNN. Let the user-specified objective function that needs to be minimized be $Obj(T, C)$, where T and C are the model training time and training cost of the CNN; Ceer can be extended to other related metrics as well, such as training throughput. For each available cloud GPU instance, Ceer estimates the T (via Eq. (2)) and C values using the trained models described in the aforementioned subsections, and recommends the GPU instance that provides the lowest estimated $Obj(T, C)$ value.

To obtain the required information that serve as input to our models, we rely on the training framework, TensorFlow (in our case). Specifically, when the CNN is deployed on TensorFlow (using the tf.Session API), the DAG of the CNN is available, which provides information on the operations and operation types involved in the CNN; we thus obtain the n (number of operations) and op_i (operation type for the i^{th} operation) values for the CNN. Likewise, the $input$ features for each operation and the number of model parameters can be obtained from TensorFlow. For other frameworks, alternatively, the target CNN can be executed for a single iteration on any machine (including a non-GPU machine) to obtain these values from logs; this approach also works for TensorFlow.

We note that when applying the models for estimating the compute time of a heavy operation, op, of the given CNN, we are assuming that op has been observed in the training data with any input data size. While insight (1) provides some justification for this assumption, it is of course possible that we encounter a heavy operation that has not been seen in training; this is especially true as new operations may be developed

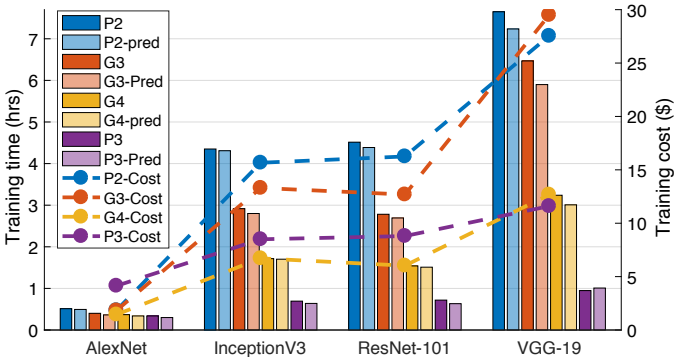

Fig. 8: Observed and predicted training time and training cost of test set CNNs on 4-GPU AWS instances.

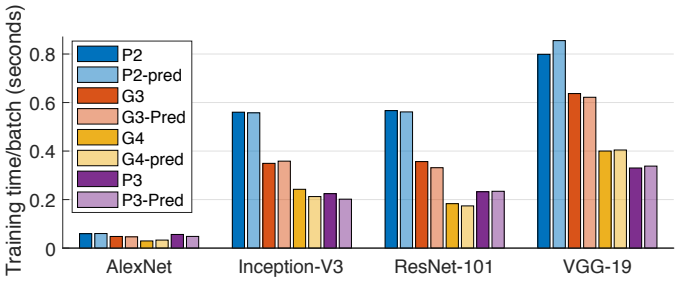

Fig. 9: Observed and predicted results for per-iteration training time given a $3/hr cost budget.

over time by researchers to support new functionalities. In such cases, Ceer will have to be updated with new training data to provide estimates for these new heavy operations. For unseen light GPU or CPU operations, we can continue to use the sample median estimates from existing training data.

V. Evaluation Results

We now present our evaluation results for Ceer. We consider several scenarios with different optimization objectives to highlight the efficacy of Ceer. We make use of single- and multi-GPU instances from AWS EC2 spanning all four GPU model types. Specifically, we employ the following 8 GPU instances (with hourly [On-Demand Pricing] rental costs in parenthesis, obtained from AWS [11]): p3.2xlarge ($3.06/hr), p2.xlarge ($0.90/hr), g4dn.2xlarge ($0.752/hr), g3s.xlarge ($0.75/hr), p3.8xlarge ($12.24/hr), p2.8xlarge ($7.20/hr), g4dn.12xlarge ($3.912/hr), and g3.16xlarge ($4.56/hr). The first four are single-GPU instances and the last four are multi-GPU instances (with at least four GPUs). Note that the design of Ceer is not specific to these instance types, and so Ceer can be applied to scenarios with different GPU model types and costs, such as other cloud provider instances.

We evaluate Ceer on the 4 test set CNNs — Inception-v3, AlexNet, ResNet-101, and VGG-19. As the input data set for the CNNs, we employ ImageNet [26], which has 1.2 Million samples, with the default batch size of 32 per GPU. We use TensorFlow r1.14 as our CNN model training framework.

To estimate training time and training cost of test set CNNs, we employ the models described in Section IV, trained on the 8 training set CNNs. Note that, in certain scenarios, we require GPU training time and cost for an instance type that is not available via AWS. For example, a 3-GPU instance of P2 GPU type is not available in AWS as it only supports 1-GPU, 8-GPU, and 16-GPU versions of the P2 GPU instance. As opposed to entirely omitting such cases, for obtaining the training time, we employ the 8-GPU instance but only use 3 of the available GPUs; for cost, we use $\frac{3}{8}$th of the rental cost of the 8-GPU instance, as a proxy.

Validation test. We start with a validation test to evaluate the accuracy of our models. The bars in Figure 8 (left y-axis) show the observed and Ceer-predicted training time for our test set CNNs when using the ImageNet [26] data set (1.2 Million samples, with default batch size of 32 per GPU) on all 4 GPU model types; here, we employ 4-GPU instances and use data parallelism to train over all 4 GPUs. We see that the predicted relative ranking of training time for each CNN is in perfect agreement with the observed ranking; note that the test set CNNs were not used for training our Ceer models. Across all experiments in Figure 8, the average training time prediction error is 5.4%. This low error highlights the high accuracy of our compute time models and our data parallelism overhead models. Since the training cost prediction is based off of the training time prediction, our cost prediction error is the same as the training time prediction error in all cases.

We see that the training time for a given 4-GPU instance is lowest for the (latest) P3 GPU model type. Compared to P2, G3, and G4, the P3 GPU model type reduces the training time on average by 72.4%, 62.9%, and 48.0%, respectively. Interestingly, the training cost (dots, right y-axis) is also quite low for P3. However, the lowest training cost is typically incurred by the G4 GPU model type, albeit at the expense of a 128% higher training time compared to P3 (averaged across all 4 test set CNNs). The P2 and G3 model types typically have significantly higher costs and higher training times compared to G4 and P3 models.

Hourly budget constrained scenario. We now consider a scenario where the objective is to minimize the per-iteration training time (or maximize the training throughput) given a limit on the hourly rental expenses. Figure 9 shows the observed and predicted results for per-iteration training time given an hourly budget of $3/hr. Under this hourly budget, and with the AWS pricing model [11], Ceer predicts the following optimal instance sizes for each GPU model: 3-GPU instance for P2, 3-GPU instance for G3, 3-GPU instance for G4, and 1-GPU instance for P3. For the 3-GPU P2 instance (since such an instance is not supported by AWS), we assume the cost is $\frac{3}{8}$th the cost of a basic 8-GPU P2 instance (p2.8xlarge, $7.20/hr); similarly for the other 3-GPU instances. (The hourly budget is slightly exceeded for P3, by 6 cents; we choose to allow this trivial violation since otherwise no P3 instance would meet the $3/hr budget. The budget is exceeded for G3 by 42 cents, but the 3-GPU G3 instance still has poor performance. Alternatively, we can consider the budget to be

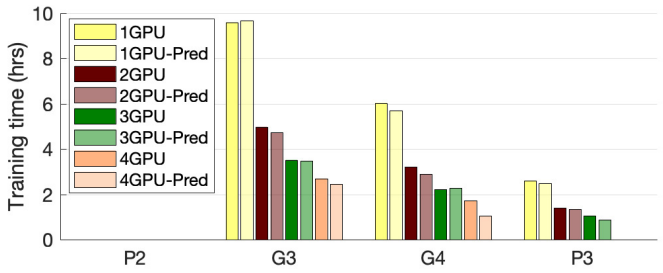

Fig. 10: Observed and predicted results for training time of ResNet-101 given a $10 total cost budget.

$3.42/hr to avoid this issue.)

We find that the optimal choice depends on the CNN type. While the P3 instance is optimal for Inception-v3 and VGG-19, the G4 instance is optimal for AlexNet and ResNet-101. This is likely because while Inception-v3 and VGG-19 have several pooling operations, AlexNet and ResNet-101 have only a few pooling operations each; recall that P3 is cost-efficient for pooling operations (see Section III-B). For all 4 CNNs, Ceer rightly predicts the relative ranking of GPU types. Further, the per-iteration training time predictions are accurate in each case, with an average prediction error of 5.6%.

By default, AWS lists the latest P3 instance types for GPU tasks [33]. If we employ this baseline strategy and pick the largest P3 instance that fits in the budget (1-GPU P3 instance), the per-iteration training time for AlexNet and ResNet-101 would increase by 91% and 27%, respectively, compared to employing Ceer (which picks the 4-GPU G4 instance).

Total budget constrained scenario. We now consider the scenario where there is a limit on the total budget that can be spent over the training time of the CNN; in this case, we are interested in determining the GPU instance type that provides the minimum training time while not exceeding the budget. Figure 10 shows the observed and predicted results for training time of ImageNet data set under ResNet-101 given a total budget of $10. Under this budget, we find that the 4-GPU instance of P3 and all P2 instances (with 1–4 GPUs) are unable to complete the training within the $10 cost. Fortunately, Ceer accurately predicts this budget violation.

Among the remaining choices, we find that the 3-GPU P3 instance provides the lowest training time. With its high prediction accuracy (5.9% error, on average, for the experiments in Figure 10), Ceer rightly predicts the 3-GPU P3 instance to be optimal. Note that picking the cheapest feasible instance (1-GPU G3 instance) and training the data set on this instance would result in a 9.1× higher training time when compared to the training time under the Ceer-predicted instance.

Budget minimization scenario. We now consider the interesting scenario where the objective is to minimize the rental cost incurred to complete model training over a given data set without any required performance target. Figure 11 shows the observed and predicted costs for training Inception-v3 using the ImageNet data set. We see that the 1-GPU G4 instance has the lowest training cost. Again, with its high prediction

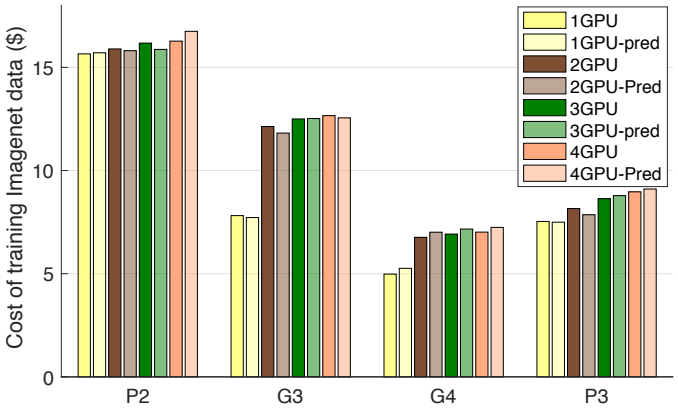

Fig. 11: Observed and predicted results for total cost of Inception-v3 training.

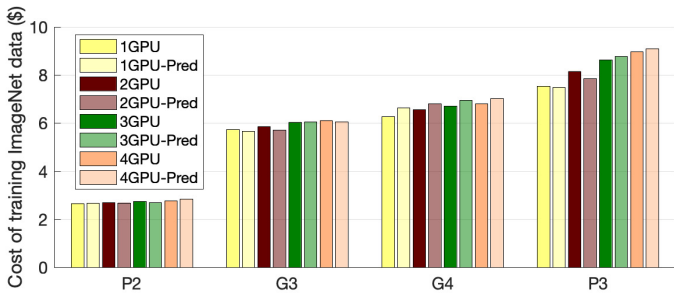

Fig. 12: Observed and predicted results for total cost of Inception-v3 training using market prices for GPU.

accuracy (2.1% cost prediction error, on average, for the experiments shown in Figure 11), Ceer rightly predicts this instance to be the optimal. Note that picking the cheapest instance (the 1-GPU G3 instance) or the most powerful instance (the 4-GPU P3 instance) would result in 1.6× and 1.8× higher training cost, respectively, when compared to the cost under the Ceer-predicted instance.

Budget minimization with commodity GPU prices ratio. For AWS, we believe that the rental cost of some of the older-generation GPU instances are not representative of their market price, likely because they have not been updated given the focus on newer GPUs. For a single GPU, the P3 (NVIDIA Tesla V100), G4 (NVIDIA T4 Tensor Core), G3 (NVIDIA Tesla M60), and P2 (NVIDIA K80) model type GPUs have normalized market price ratios of 1:0.31:0.18:0.05 (based on server costs from amazon.com [34], [35], [36], [37]); the corresponding AWS price ratio is about 1:0.25:0.25:0.29. As a result, the older-generation GPU instances are at a disadvantage in terms of the cost-performance tradeoff under AWS prices. In this scenario, we consider the AWS instances with updated rental prices to reflect their GPU market rates. Using the market price ratio, we consider the hourly rental costs for a single-GPU instance of GPU type P3, G4, G3, and P2 to be $3.06 (actual AWS cost), $0.95, $0.55, and $0.15, respectively. We consider the cost of multi-GPU instances to be linearly scaled up versions of these costs.

Figure 12 shows the observed and predicted costs for training Inception-v3 using the ImageNet data set, but with market prices for the GPU instances. We see a stark contrast between the results of Figure 11 and Figure 12, suggesting that the GPU instance prices significantly impact the training cost. In this case, the lowest training cost is provided by the 1-GPU P2 instance; Ceer again rightly predicts this optimal instance, and has a low cost prediction error of 2.1%, on average, for the experiments shown in Figure 12. If we instead employ the 1-GPU G4 instance (which was optimal for Figure 11), the training cost increases by 2.4×, compared to the cost of the Ceer-predicted optimal instance.

VI. Limitations

By design, Ceer should be able to accurately predict the training time and cost of an arbitrary CNN on different cloud GPU instance types. However, given the range of experiments conducted in this paper, Ceer does have some limitations. First, as mentioned in Section IV-D, Ceer cannot predict (without retraining) the training time of a CNN that includes a heavy operation that has not been observed during training. Second, all our experiments assume that the multiple GPUs are part of the same host; with GPUs spread across hosts, the communication model of Ceer will have to be retrained. Third, while the simple yet accurate additive model of Ceer (see Section IV) works well for single GPU execution or data-parallel execution of CNNs, it may not be accurate for model-parallel training because of the overlap of compute and communication operations. Fourth, all the experiments in this paper only consider CNNs. It will be interesting to see how Ceer performs on other types of DNNs, such as Recurrent Neural Nets (RNNs) or Transformer models for Natural Language Processing. Finally, Ceer is currently designed to work with TensorFlow and may not provide accurate estimations for other machine learning frameworks, such as PyTorch [38].

VII. Prior Work

Empirical studies on DNN performance. Ren et al. [39] analyze the performance of different DNNs across platforms, including cloud instances. However, the authors only focus on profiling the communication overhead involved when parallelizing the model training. Mojumder et al. [40] analyze the overhead associated with different synchronization strategies for data parallel training. Zhu et al. [41] present a DNN benchmark and performance analysis tool for analyzing the performance of a given DNN across multiple devices and multiple parallel settings. While the tool can provide useful insights on performance bottlenecks, it cannot predict the performance of a new CNN.

Approaches for training time prediction. Most of the prior work on predicting the training time of DNNs focuses on higher-level components of the DNN, such as layers (composed of several operations) or the per-iteration training time. Gianniti et al. [4] present a simple linear regression based approach for predicting execution times for a CNN on a given underlying GPU device. However, the authors focus on layer-level modeling, and ignore small operations and CPU operations, resulting in prediction errors as high as 22%. As we discuss in Section IV, ignoring light operations or CPU operations can significantly hurt prediction accuracy.

Cai et al. [17] model the per-iteration time based on polynomial regression of popular layers. Justus et al. [20] model the training time based on a deep learning model of individual layers. Neurosurgeon [42] leverages regression modeling to predict the execution time of popular layers. Cai et al. and Justus et al. focus on a single-GPU instance and do not account for communications costs; as discussed in Section IV-A, ignoring the communication overhead for a single GPU instance can result in high prediction error. Neurosurgeon does account for some communication overhead, but not between GPUs, and so their model does not extend to multi-GPU instances.

PALEO [43] predicts per-iteration time based on a linear model of the number of floating-point operations in each iteration; however, this model does not capture the impact of communication overhead or input data sizes on per-iteration training time. FastDeepIoT [19] profiles different DNN operations on mobile devices to build a tree-structured-linear regression model for execution time of different DNNs on that device. Lu et al. [5] propose a solution for predicting the memory and compute efficiency of different CNNs on different mobile devices. The models developed by FastDeepIoT and Lu et al. [5] are specific to mobile devices and do not extend to cloud GPU instances or to multi-GPU devices.

The FlexFlow [18] work develops a simulator to predict the execution times of a DNN model under different parallelization strategies. The simulator typically works by running the DNN for a few iterations to collect data on the DNN, and then simulating different strategies. In doing so, the simulator relies on having prior observations for an operation for every input data size encountered, thus making it difficult to employ the simulator to predict training time for a new CNN.

Finally, prior work has shown that, for specific CNNs, a few compute operations make up a large percentage of the training time [44], [18]; for example, Liu et al. [32] find that operations like conv2DBackpropFilter and Conv2DBackpropInputs contribute to 40% of the training time for VGG-19 and AlexNet. However, only considering a subset of the operations can result in high prediction error, as discussed in Section IV-B.

Other parallelization approaches for CNN training. While Ceer focuses on data parallelism for scaling CNN training across GPUs, there are other parallelization techniques that can be employed. Model parallelism partitions the CNN graph into subgraphs, with each subgraph being placed on a different GPU [45], [46], [30]. Prior works have also explored pipeline parallelism [30], [31] to better overlap communication with computation when employing model parallelism. Finally, recent work [18] has proposed operation-level parallelism to reduce the compute times of individual operations by leveraging multiple GPUs. For these parallelization techniques, the training time per iteration cannot be easily obtained from the per-GPU compute times because of the dependencies between

tasks across GPUs; we will investigate training time modeling under these techniques as part of future work.

VIII. Conclusion

With the proliferation of AI and ML in almost all fields of science and technology, GPU-equipped machines are in very high demand for training neural networks for inference and other learning tasks. Given the high cost of GPUs, practitioners are increasingly turning to cloud-offered GPU resources to train their models. However, the range of GPU model types offered by cloud providers, along with their varying price points, makes it difficult for practitioners to determine the best choice of GPU resources to use. Further complicating the decision is the fact that cloud providers offer instances with varying number of GPUs on them, with proportionally higher costs; however, leveraging more GPUs need not result in linearly increased performance.

This paper presents Ceer, a data- and model-driven approach to estimate the model training time and training cost of arbitrary CNNs across different GPU instance types. Ceer accurately estimates the relative ranking of training time and cost on different GPU models by relying on the key insights that a small number of unique operation types contribute to most of the CNN training time and that the compute time of these operations, for a given input size, has very low variability. To increase prediction accuracy, Ceer employs the sample median estimate for light GPU operations and CPU operations, resulting in a training time (and cost) prediction accuracy of more than 94%. Ceer is also able to accurately predict how the training time scales with the number of GPUs by leveraging the nearly linear relationship between communication overhead under data parallelism and the number of model parameters. Evaluation results on AWS EC2 across various CNN scenarios show that Ceer can accurately predict the optimal GPU configuration that minimizes a user-specified objective function of training time and training cost.

Acknowledgment

This work was supported by NSF CNS grants 1717588 and 1750109, and the AWS Cloud Credits for Research program.

References

[1] G. Hinton, L. Deng, D. Yu, G. Dahl, A.-R. Mohamed, N. Jaitly, A. Senior, V. Vanhoucke, P. Nguyen, B. Kingsbury *et al.*, "Deep neural networks for acoustic modeling in speech recognition," *IEEE Signal processing magazine*, vol. 29, 2012.

[2] N. Kalchbrenner, E. Grefenstette, and P. Blunsom, "A convolutional neural network for modelling sentences," in *52nd Annual Meeting of the Association for Computational Linguistics*. Association for Computational Linguistics, 2014.

[3] 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.

[4] E. Gianniti, L. Zhang, and D. Ardagna, "Performance prediction of gpu-based deep learning applications," in *2018 30th International Symposium on Computer Architecture and High Performance Computing (SBAC-PAD)*. IEEE, 2018, pp. 167–170.

[5] Z. Lu, S. Rallapalli, K. Chan, and T. La Porta, "Modeling the resource requirements of convolutional neural networks on mobile devices," in *Proceedings of the 25th ACM international conference on Multimedia*, 2017, pp. 1663–1671.

[6] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard *et al.*, "Tensorflow: A system for large-scale machine learning," in *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*, 2016, pp. 265–283.

[7] THINKMAKE, "Thinkmate gpx xt4-24s1-4nvlink," https://www.thinkmate.com/system/gpx-xt4-24s1-4nvlink.

[8] Amazon Web Services, Inc., "Amazon EC2 P3 Instances," https://aws.amazon.com/ec2/instance-types/p3.

[9] Google Cloud, "Cloud gpus," https://cloud.google.com/gpu".

[10] Microsoft Azure, "Gpu optimized virtual machine sizes," https://docs.microsoft.com/en-us/azure/virtual-machines/sizes-gpu.

[11] Amazon Web Services, Inc., "Amazon EC2 Pricing," https://aws.amazon.com/ec2/pricing/on-demand.

[12] N. Strom, "Scalable distributed dnn training using commodity gpu cloud computing," in *Sixteenth Annual Conference of the International Speech Communication Association*, 2015.

[13] 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.

[14] J. Hu, L. Shen, and G. Sun, "Squeeze-and-excitation networks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 7132–7141.

[15] T. Ridnik, H. Lawen, A. Noy, and I. Friedman, "Tresnet: High performance gpu-dedicated architecture," *CoRR*, vol. abs/2003.13630, 2020. [Online]. Available: https://arxiv.org/abs/2003.13630

[16] S. Gupta, W. Zhang, and F. Wang, "Model accuracy and runtime tradeoff in distributed deep learning: A systematic study," in *2016 IEEE 16th International Conference on Data Mining (ICDM)*. IEEE, 2016, pp. 171–180.

[17] E. Cai, D. Juan, D. Stamoulis, and D. Marculescu, "Neuralpower: Predict and deploy energy-efficient convolutional neural networks," in *Proceedings of the 2017 Asian Conference on Machine Learning (ACML)*, Seoul, South Korea, 2017.

[18] Z. Jia, M. Zaharia, and A. Aiken, "Beyond data and model parallelism for deep neural networks," *arXiv preprint arXiv:1807.05358*, 2018.

[19] S. Yao, Y. Zhao, H. Shao, S. Liu, D. Liu, L. Su, and T. Abdelzaher, "Fastdeepiot: Towards understanding and optimizing neural network execution time on mobile and embedded devices," in *Proceedings of the 16th ACM Conference on Embedded Networked Sensor Systems*, 2018, pp. 278–291.

[20] D. Justus, J. Brennan, S. Bonner, and A. S. McGough, "Predicting the computational cost of deep learning models," in *Proceedings of the IEEE International Conference on Big Data*, Seattle, WA, USA, 2018, pp. 3873–3882.

[21] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016, http://www.deeplearningbook.org.

[22] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, "Rethinking the inception architecture for computer vision," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 2818–2826.

[23] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.

[24] ZDNet, "Top cloud providers in 2020: Aws, microsoft azure, and google cloud, hybrid, saas players," https://www.zdnet.com/article/the-top-cloud-providers-of-2020-aws-microsoft-azure-google-cloud-hybrid-saas.

[25] ParkMyCloud, "Aws vs azure vs google cloud market share 2020: What the latest data shows," https://www.parkmycloud.com/blog/aws-vs-azure-vs-google-cloud-market-share.

[26] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *Proceedings of the 2009 IEEE conference on computer vision and pattern recognition*, ser. CVPR'09, Miami, FL, USA, 2009, pp. 248–255.

[27] J. Dean, G. Corrado, R. Monga, K. Chen, M. Devin, M. Mao, A. Senior, P. Tucker, K. Yang, Q. V. Le *et al.*, "Large scale distributed deep networks," in *Advances in neural information processing systems*, 2012, pp. 1223–1231.

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

[29] T. Chilimbi, Y. Suzue, J. Apacible, and K. Kalyanaraman, "Project adam: Building an efficient and scalable deep learning training system," in *11th USENIX Symposium on Operating Systems Design and Implementation (OSDI 14)*, 2014, pp. 571–582.

[30] Y. Huang, Y. Cheng, D. Chen, H. Lee, J. Ngiam, Q. V. Le, and Z. Chen, "Gpipe: Efficient training of giant neural networks using pipeline parallelism," *arXiv preprint arXiv:1811.06965*, 2018.

[31] A. Harlap, D. Narayanan, A. Phanishayee, V. Seshadri, G. R. Ganger, and P. B. Gibbons, "PipeDream: Pipeline Parallelism for DNN Training," in *Proceedings of the 1st Conference on Systems and Machine Learning (SysML)*, Stanford, CA, USA, 2018.

[32] J. Liu, H. Zhao, M. A. Ogleari, D. Li, and J. Zhao, "Processing-in-Memory for Energy-Efficient Neural Network Training: A Heterogeneous Approach," in *Proceedings of the 51st Annual IEEE/ACM International Symposium on Microarchitecture*, ser. MICRO-51, Fukuoka, Japan, 2018, pp. 655–668.

[33] Amazon Web Services, Inc., "Amazon EC2 Instance Types," https://aws.amazon.com/ec2/instance-types.

[34] amazon.com, "Supermicro tesla k80 graphic card - 2 gpus - 562 mhz core - 875 mhz boost clock - 24 gb gddr5 sdram - pci express 3.0 x16 - dual aoc-gpu-nvk80," https://www.amazon.com/Supermicro-Tesla-Graphic-Card-AOC-GPU-NVK80/dp/B014EC0LN0.

[35] amazon.com, "Nvidia tesla m60 16gb server gpu accelerator processing card hp 803273-001," https://www.amazon.com/NVIDIA-Accelerator-Processing-HP-803273-001/dp/B01FL56SZY.

[36] amazon.com, "Hp r0w29a tesla t4 graphic card - 1 gpus - 16 gb," https://www.amazon.com/HP-R0W29A-Tesla-Graphic-Card/dp/B07PGY6QPT.

[37] amazon.com, "Hp nvidia tesla v100 16gb pcie x16 876340-001 876908-001 q2n68a," https://www.amazon.com/HP-nVidia-876340-001-876908-001-Q2N68A/dp/B07DJYQ5W7.

[38] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga *et al.*, "Pytorch: An imperative style, high-performance deep learning library," in *Advances in neural information processing systems*, 2019, pp. 8026–8037.

[39] Y. Ren, S. Yoo, and A. Hoisie, "Performance analysis of deep learning workloads on leading-edge systems," in *2019 IEEE/ACM Performance Modeling, Benchmarking and Simulation of High Performance Computer Systems (PMBS)*. IEEE, 2019, pp. 103–113.

[40] S. A. Mojumder, M. S. Louis, Y. Sun, A. K. Ziabari, J. L. Abellán, J. Kim, D. Kaeli, and A. Joshi, "Profiling dnn workloads on a volta-based dgx-1 system," in *2018 IEEE International Symposium on Workload Characterization (IISWC)*. IEEE, 2018, pp. 122–133.

[41] H. Zhu, M. Akrout, B. Zheng, A. Pelegris, A. Jayarajan, A. Phanishayee, B. Schroeder, and G. Pekhimenko, "Benchmarking and analyzing deep neural network training," in *2018 IEEE International Symposium on Workload Characterization (IISWC)*. IEEE, 2018, pp. 88–100.

[42] Y. Kang, J. Hauswald, C. Gao, A. Rovinski, T. Mudge, J. Mars, and L. Tang, "Neurosurgeon: Collaborative intelligence between the cloud and mobile edge," *SIGARCH Comput. Archit. News*, vol. 45, no. 1, p. 615–629, Apr. 2017.

[43] H. Qi, E. R. Sparks, and A. Talwalkar, "Paleo: A performance model for deep neural networks," in *Proceedings of the 5th International Conference on Learning Representations, ICLR*, 2017.

[44] J. Liu, D. Li, G. Kestor, and J. Vetter, "Runtime concurrency control and operation scheduling for high performance neural network training," in *Proceedings of the 2019 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, Rio de Janeiro, Brazil, 2019, pp. 188–199.

[45] A. Mirhoseini, H. Pham, Q. V. Le, B. Steiner, R. Larsen, Y. Zhou, N. Kumar, M. Norouzi, S. Bengio, and J. Dean, "Device placement optimization with reinforcement learning," in *Proceedings of the 34th International Conference on Machine Learning-Volume 70*. JMLR. org, 2017, pp. 2430–2439.

[46] A. Mirhoseini, A. Goldie, H. Pham, B. Steiner, Q. V. Le, and J. Dean, "A hierarchical model for device placement," 2018.