PROFET: PROFiling-based CNN Training Latency ProphET for GPU Cloud Instances

Sungjae Lee, Yoonseo Hur, Subin Park, Kyungyong Lee
Department of Computer Science, Kookmin University, Seoul, South Korea
{sungjae, yoonseo, subean, leeky}@kookmin.ac.kr

Abstract—Training a Convolutional Neural Network (CNN) model typically requires significant computing power, and cloud computing resources are widely used as a training environment. However, it is difficult for CNN algorithm developers to keep up with system updates and apply them to their training environment due to quickly evolving cloud services. Thus, it is important for cloud computing service vendors to design and deliver an optimal training environment for various training tasks to lessen system operation management overhead of algorithm developers. To achieve the goal, we propose PROFET, which can predict the training latency of arbitrary CNN implementation on various Graphical Processing Unit (GPU) devices to develop a cost-effective and time-efficient training cloud environment. Different from the previous training latency prediction work, PROFET does not rely on the implementation details of the CNN architecture, and it is suitable for use in a public cloud environment. Thorough evaluations reveal the superior prediction accuracy of PROFET compared to the state-of-the-art related work, and the demonstration service presents the practicality of the proposed system.

Index Terms—CNN, training, latency, prediction, GPU, cloud

I. INTRODUCTION

Deep Neural Network (DNN) has paved the way for the application of artificial intelligence algorithms in many fields, including image recognition and natural language processing. The success of deep learning is due to the advancement in DNN algorithms [1], programming interfaces and platforms [2]–[4], and optimized hardware for deep learning, such as the GPU and Tensor Processing Unit (TPU) [5].

Among many DNN algorithms, Convolutional Neural Network (CNN) [6]–[8] is widely used for various applications, such as object detection for self-driving cars and image recognition. The complex internal structure of CNN implementations requires massive computing power and parallelism, particularly during the training step. Depending on the model complexity, training a CNN model can take a few weeks or months [9], [10], and building a cost-efficient and fast processing environment is crucial to increase the productivity of CNN algorithm developers. In addition, GPU devices can provide massive parallelism, but building a GPU cluster can be prohibitively expensive. Users can easily build a CNN model without purchasing GPU devices by using cloud resources. Training jobs may occur on occasion, and using GPU instances only when needed can result in cost savings.

The pace of cloud computing innovation is very fast, with new types of cloud services and instances being released on a regular basis by public vendors. It is difficult for deep learning algorithm developers to keep up with updates from public cloud vendors and understand the impact of the updates in order to apply them to their deep learning training pipeline. As a result, it is ideal if cloud service vendors provide an optimized deep learning environment natively so that algorithm developers can focus on core application development.

The Platform-as-a-Service (PaaS) cloud service model promoted as reducing development platform management overhead, allowing programmers to focus on critical software development tasks. Current deep learning pipeline platform services provided by public cloud vendors, such as AWS SageMaker and Google Cloud Datalab, are far from optimal because users must manually select cloud instance types and the number of instances to train a DNN model. Furthermore, an optimal DNN environment can vary across model architectures, dataset sizes, and model configurations, making deep learning platform management more difficult.

To understand the performance characteristics of the various CNN algorithm training latency on different public cloud GPU instances, we measured the training time on AWS EC2 instances that are equipped with GPU devices (Table I), and the result is presented in Figure 1. Different CNN model architectures could result in a five-fold performance difference between the best and worst-performing GPU instance types. Despite such stark performance differences when training a CNN model on GPU cloud instances, algorithm developers rarely understand such performance characteristics and miss opportunities to optimize training environments. In addition, imposing training environment operation burdens on DNN algorithm developers might be too much overhead for them, as they are not generally system experts. Thus, the cloud service vendor should prepare an optimal training environment that reflects algorithm developers’ implementation characteristics.

Habitat [11], Paleo [12], NeuralPower [13], and MLPredict [14] proposed algorithms to predict the training time for various CNN algorithms on different GPU devices. The systems use the internal architecture of a CNN implementation and GPU device characteristics as input features. We call such approaches white-box methods. Though they envisioned the feasibility of predicting the training time of CNN implementations on arbitrary GPU devices, the white-box approach can be challenging to apply in an environment where a deep learning algorithm developer and a DNN development platform provider, such as a public cloud vendor, are not identical. In public cloud services, to provide an optimal DNN training
environment using a white-box approach, a service provider should know details of the model architecture. However, it is unlikely that algorithm developers would be willing to share the source code, which is generally confidential and is a private asset of an organization.

To overcome the shortcomings of the previous works, we propose PROFET, which aims to predict the training time for arbitrary CNN implementations without revealing the internal model architecture. Hiding the model architecture in a training latency prediction model makes the proposed system appropriate for a public cloud system where algorithm developers and development platform maintainers are different. To meet the goal, PROFET uses abstracted profiling information from CNN training as prediction model input and proposes novel heuristics of median-ensemble modeling to enhance prediction accuracy. PROFET can predict the CNN model training time on diverse GPU devices to support CNN training scenarios on the cloud GPU instances. A thorough evaluation of PROFET reveals that it outperforms the state-of-the-art algorithms, Habitat [11], Paleo [12], and MLPredict [14], improving prediction accuracy by 32%, 68%, and 82%, respectively.

In summary, the major contributions are as follows.

- Envisioning the importance of black-box performance estimation of the CNN on a public cloud
- Unique median-ensemble modeling to predict training latency across distinct GPU devices
- Publicly available artifacts and a web service to enhance the development of a deep learning system

II. TRAINING CNN ON CLOUD

Training a model can take weeks, depending on the internal architecture of the CNN implementation [10], [15], [16]. Various configurations influence CNN model training time, and it is critical to understand the performance diversity of CNN training to build an optimal training environment.

A. Overview of CNN

CNN algorithms are commonly used to analyze visual representations from a given input dataset, which typically consists of images or videos. The most common type of CNN model is built with input and output layers connected by a series of hidden layers. Each hidden layer includes a wide range of operations, such as convolution, activation, and pooling. The operations extract information from the previous layer

![Relative Latency](image)

Table I: Specification of different GPU Instances on AWS

| Instance Family | G3s | G4(dn) | P2 | P3 |
|-----------------|-----|--------|----|----|
| GPU Model       | M60 | T4     | K80| V100|
| GPU Core        | 2048| 2560   | 2496| 5120|
| GPU Clock(MHz)  | 1178| 1590   | 875 | 1380|
| TFLOPS(FP32)    | 4,825| 8,141 | 4,113| 14,13|
| Released Year   | 2017| 2019   | 2016| 2017|
| Price($/hr)     | 0.75| 0.526  | 0.9 | 3.06|

C. Understanding CNN Training Performance

The diversity of CNN model, configurations, and cloud instances can result in significant differences in model training time. To represent latency and cost difference under various settings, we conducted thorough experiments. Figure 1 presents the mini-batch training time of LeNet5 [6] and AlexNet [8] on various GPU cloud instances presented in Table I. The latencies are represented with bars in the primary vertical axis. The secondary vertical axis displays the relative cost of completing a given training workload, with the values represented by star marks. They are normalized to the least value in each workload.

PROFET Service and Artifacts: http://profet.ddps.cloud
For each workload, the cloud instances are shown in the order of g3s, g4dn, p2, and p3. For LeNet5, the g4dn is the fastest, while the p3 is the fastest for AlexNet training. Comparing the best and worst-performing instance types, the latency of LeNet5 is less than twice times (g4dn and p2), but that of AlexNet is close to ten times (p3 and p2). Regarding the cost, g4dn incurs the least cost for both workloads, but choosing p3 instance can be more beneficial for AlexNet because its training latency is about one-third with only twice more cost than g4dn.

Figure 1 presents drastic CNN training latency variations on diverse cloud instances with GPU devices under different model architectures. Considering the continuous release of new cloud services, resources, and pricing mechanisms [19]–[21], it is challenging for CNN algorithm developers to follow new updates and timely apply them to their training environment. Furthermore, it is very cumbersome to try every instance type that might need custom device library setup to check how a CNN model performs. Therefore, a guidance of estimated performance of custom CNN algorithm implementations on diverse cloud instance types is mandatory to help developers build an optimal training environment and focus on core application development.

III. PROFET: Modeling Training Time

The goal of PROFET is to predict the training time for arbitrary CNN implementations on various GPU instances provided by a public cloud service vendor with minimal exposure to implementation details. Minimally exposing the implementation details is important, especially in a public cloud environment. Setting up a CNN development environment using GPUs requires considerable system operation effort, which can be quite challenging for an algorithm developer. Thus, the responsibility of maintaining and operating an optimal development environment should be imposed on the cloud service vendor, which agrees with the recent cloud computing evolution direction [22] represented by the serverless computing [23], [24]. To allow public cloud vendors to provide an optimal training environment for arbitrary CNN implementations, CNN model characteristics should be provided while hiding the detailed internal architecture, and PROFET achieves this goal using abstracted operation information.

To build a prediction model, a set of feature vectors which we denote as \( \mathbf{X} \) is generated from offline experiments. \( \mathbf{X} \) is composed of \( N \) workloads. Each workload return a feature vector with a dimension of \( D \). Thus, the dimension of \( \mathbf{X} \) is \( N \times D \). We denote each workload scenario as \( x_i, i = 1 : N \). To note \( j \)-th feature of workload \( i \), where \( i = 1 : N, j = 1 : D \), we use \( x_{ij} \).

To generate diverse CNN workloads, we variate CNN training scenarios with respect to the instance types \( G \), model architectures \( M \), batch sizes \( B \), and input image pixel sizes \( P \). For \( G \), we assume GPU cloud instances provided by AWS, \( G \ni \{g3s, g4dn, p2, p3\} \). For models, \( M \), we used well-known CNN models in literature, \( M \ni \{\text{AlexNet}, \text{LeNet5}, \text{InceptionV3}, \text{InceptionResNetV2}, \text{MobileNetV2}, \text{MNIST}, \text{CIFAR10}, \text{ResNetSmall}, \text{ResNet18}, \text{ResNet34}, \text{ResNet50}, \text{VGG11}, \text{VGG13}, \text{VGG16}, \text{VGG19}\} \). We used five batch sizes, \( B \ni \{32, 512, 112, 112\} \), and five input image pixel sizes, \( P \ni \{32 \times 32, 64 \times 64, 128 \times 128, 224 \times 224, 256 \times 256\} \). To generate images with different pixel sizes, we used the Numpy library. All workload scenarios can be generated by conducting Cartesian product in the four dimensions \( (G \times M \times B \times P) \). All cases of \( G \times M \times B \times P \) cannot be completed due to hardware or model constraints. Filtering out inexecutable cases, we finalize 1228 cases which becomes the cardinality of input dataset, which is \( N \). Each workload \( x_i \) is a vector with a length \( D \) that is returned by the underlying profiler. Among the operations provided by TensorFlow Profiler [25], we generate 65 aggregated high-level operations from 1228 executable workloads. Thus, the dimension of \( \mathbf{X} \) is \( 1228 \times 65 \).

A. Extracting CNN Characteristics on Cloud

A PaaS cloud model frees CNN algorithm developers from operating cloud infrastructures and DNN development platforms because a cloud service provider is responsible for maintaining them. To provide efficient deep learning platforms, cloud vendors should understand distinct workload characteristics submitted by clients. However, in a PaaS model, most CNN algorithm developers are not willing to share source codes or internal architectures of a developed model. Thus, previous works which reference internal model architectures to predict DNN training latency [12]–[14] cannot be applied directly on a PaaS environment. To predict training latency of arbitrary CNN implementations on a public PaaS model, PROFET should be able to characterize a workload while minimally disclosing the internal model architecture.

Other than referencing source code to characterize CNN workloads, PROFET proposes to use performance metrics generated during a training phase provided by a profiler of DNN programming platforms, such as TensorFlow [3], MXNet [26], Torch [2], and Theano [4]. Though information from metrics of each platform differs slightly, they provide response times for core DNN-specific operations in common.

To better understand performance metrics provided by a DNN development platform, we show an example of a CNN model profiling outcome generated by TensorFlow Profiler [25] in Figure 2. A profiling output contains Operation, Operation

![Fig. 2: An example of profiling data generated from AlexNet model training using TensorFlow profiler](image-url)

| Operation | Operation Details | Time (ms) |
|-----------|-------------------|-----------|
| conv2d_0  | [32, 96, 224, 244]| 610580 KB |
| conv2d_1  | [32, 256, 112, 112]| 104104 KB |
| conv2d_2  | [32, 512, 56, 56]| 220460 KB |
| ReLU      | activation_0      | -         | 11        |
| ReLU      | activation_1      | -         | -         | 3         |
| ReLU      | activation_2      | -         | -         | 3         |
| MaxPool   | max_pooling2d_0   | [32, 96, 112]| 154140 KB |
| MaxPool   | max_pooling2d_1   | [32, 128, 64, 64]| 120760 KB |
| MaxPool   | max_pooling2d_2   | [32, 128, 20, 20]| 100660 KB |
| ...       | ...               | ...       | ...       |

\[ \underbrace{\text{and more operations}}_{\text{Conv2D, BatchNorm, Flatten, ...}} \]
detail, and Latency fields. The operation field indicates a method name used in a source code which is specified by a development platform. The operation details field contains rich information of a CNN model which includes method name and its layer location, input and output tensor sizes, memory usage, and many more. The information contains internal architecture of custom implementation, and a model can be assembled using the information. Of the profiling outcome, PROFET proposes to use the operation field and the corresponding aggregated time field as features to represent CNN workloads and make a model to predict training time of arbitrary CNN implementations. With the high level information about operations, Hafeez et al. proved that profiled outcome can well represent characteristics of arbitrary CNN models [27]. Furthermore, hiding operation details can relieve CNN algorithm developers when they share the workload characters with public cloud PaaS vendors which can offer an optimal development environment for any kinds of CNN workloads. In summary, PROFET adopts a black-box approach by using high-level expression of CNN workloads without using internal model architectures which is contrary to the white-box approach adopted in previous work [12]–[14], and such black-box design fits very well with the public cloud environment where the service providers and consumers differ.

Using the operation names and aggregated latencies as features constructs input vectors, \( \mathbf{X} \), and we define output vectors, \( \mathbf{Y} \), as a corresponding batch latencies of the input vectors. Formally speaking, for an arbitrary CNN workload profiling features, \( x_i, y_i \) is a scalar value which represents a measured batch latency of workload \( i \) where \( i = 1 : N \). To note a value of specific feature, \( j \), of model \( i \), we use \( x_{ij} \), where \( i = 1 : N, j = 1 : D \), which is a scalar value. Creating feature vectors, \( \mathbf{X} \), from a deep learning platform incur non-negligible extra overhead that can impact batch latency, \( \mathbf{Y} \). In our off-line experiments, about 20%–30% larger batch latency is measured when we enabled profiling. To remove the impact from the profiling overhead, we conducted two sets of experiments with a workload, \( i \), one with profiling enabled, another without profiling enabled. In the experiments with profiling, we gather \( \mathbf{X} \). To get an accurate value of \( \mathbf{Y} \), we measured the batch latency without enabling profiling. This procedure of separately generating \( \mathbf{X} \) and \( \mathbf{Y} \) is identical when a new CNN workload scenario is predicted with PROFET; an user enables profiling to get a feature vector to get predicted batch training latency without profiling enabled.

**B. Modeling CNN Performance on Cloud**

PROFET predicts the training latency of arbitrary CNN implementations (\( M \)) on different GPU-based instance types (\( G \)), batch sizes (\( B \)), and input image pixel sizes (\( P \)). To formally express input datasets, \( x_i \), in a finer-grained way, we use the lowercase letter of each category as a sub-script, \( x_{mbp} = \{(m, g, b, p) : m \in M, g \in G, b \in B, p \in P \} \). For example, to specify input datasets of an arbitrary model (\( m \)), batch size (\( b \)), and input pixel size (\( p \)) for all the possible GPU instances in \( G \), we use \( x_{mbp} \) which excludes \( g \) from the sub-script, and the cardinality of \( x_{mbp} \) equals the size of \( |G| \), which means four instance types with GPU devices used in this proposed work.

The cross-instance performance model predicts the training latency of an arbitrary workload of \( x_{mbp} \) on various instance types. This model requires a profiled feature set of a workload, \( x_{mbp} \), on an instance type which we call an anchor instance, \( g_a \). Using the profiling outcome from \( g_a \), PROFET predicts the training latency on a target instance type, \( g_t \), where \( g_a \neq g_t \), and \( g_a, g_t \in G \), of workload \( x_{mbp} \). Let us define a training latency prediction model from \( g_a \) to \( g_t \), as \( f_{g_a \rightarrow g_t} \). The training dataset of the function is defined as \( D_{g_a \rightarrow g_t} \), which is defined as follows.

\[
D_{g_a \rightarrow g_t} \triangleq \{(x_{mbp}|G=g_a, y_{mbp}(G=g_t)|mbp \in (M \cup B \cup P)\}
\]

For an arbitrary workload of \( mbp \), an input feature vector \( x \) is returned from a profiler after executing on an anchor instance, \( g_a \). The output of the model, \( y \), is a scalar value that is a batch latency for the same workload, \( mbp \), executed on a target instance, \( g_t \).

With training dataset, \( D_{g_a \rightarrow g_t} \), PROFET builds a prediction model using an ensemble algorithm [28] with a median operator [29]. In machine learning, ensemble modeling combines multiple individual weak models to have higher prediction accuracy. The bootstrap aggregating (bagging) [28] trains multiple models using the same set of train dataset and allocate weights for each model to come up with the final prediction. Lang et al. [29] proposed a median-based bagging algorithm that adopts the median predicted values among multiple predictive values. According to the authors, using the median value improves model accuracy by removing noise in real-world signal processing applications. For ensembling, PROFET builds three independent prediction models of a DNN, random forest [30], and linear regressor. We build a DNN model of \( 128 \times 64 \times 32 \times 16 \times 1 \) dense layer architecture with ReLU activation [31] in each layer while minimizing the combined loss of Mean Absolute Percentage Error (MAPE) and Root Mean Squared Error (RMSE) with the Adam optimizer [32]. To build random forest and linear regression models, we use Python’s SciKit-Learn library [33] with the default hyper-parameters provided by the library.

Figure 3 explains the overall procedure of cross-instance batch latency prediction. The left side of the figure (phase 1) expresses the generation of the training dataset. In the step, all the workloads are executed on all the GPU instance types in \( G \), and profiling features with the corresponding batch latencies are recorded. To use the experimental result as a model training input, we match the profiled feature (\( x_{mbp} \)) of an anchor instance (\( x_a \)) and batch latencies (\( y_{mbp} \)) of target instances (\( x_t \)). In the figure, the anchor instance type is \( g_a \), and the target instance types are \( g_4, p_2, p_3 \). It shows three training input cases; for a same profiled feature from \( g_3 \), three batch latencies from \( g_4, p_2, p_3 \) are matched as outputs. In the prediction model building step, separate models are trained per each anchor and target instance type combination using distinct training dataset of \( D_{g_a \rightarrow g_t} \). Using the unique dataset per anchor and target instance types, three models of linear regressor, random forest, DNN are trained.
for median-ensemble modeling. In the figure, three separate ensemble models are built, \( f_{g3s \rightarrow xlarge} \), \( f_{g3s \rightarrow p2xlarge} \), and \( f_{g3s \rightarrow p3xlarge} \).

1) Predicting Latency on New GPU Types: A new type of GPU is released quite often, and it is important to anticipate training latency on such new hardware which was not available during the prediction modeling. PROFET does not use hardware specifications as prediction model features as previous work did [11], [12], [14], and it cannot predict training latency for a new GPU device which was not available at a modeling phase. However, this does not become a significant issue when the new GPU device which was not available at a modeling phase.

As shown in the figure, PROFET shows the lowest error in MAPE, and RMSE with other approaches in Figure 5. The primary vertical axis represents the MAPE whose values are shown in solid gray bars. The secondary vertical axis shows the RMSE whose values are shown in the bars with the right upper diagonal pattern. For both MAPE and RMSE, lower values are better. In the horizontal axis, we show distinct prediction models of a linear regressor (Linear), a tree-based non-linear regressor (RandomForest), DNN, and the proposed ensemble algorithm (PROFET). In the Linear approach, different from others, we use batch latency measured from the anchor instance as input data. The linear prediction model is an order-1 regressor which can be implemented by a simple linear regressor. Using the three different models, the median ensemble approach of PROFET takes the three predicted values and determines the median value as the predicted latency.

To quantitatively evaluate the prediction accuracy of PROFET, we compare the MAPE, and RMSE with other approaches in Figure 5. The primary vertical axis represents the MAPE whose values are shown in solid gray bars. The secondary vertical axis shows the RMSE whose values are shown in the bars with the right upper diagonal pattern. For both MAPE and RMSE, lower values are better. In the horizontal axis, we show distinct prediction models of a linear regressor (Linear), a tree-based non-linear regressor (RandomForest), DNN, and the proposed ensemble algorithm (PROFET). In the Linear approach, different from others, we use batch latency measured from the anchor instance as input data. The linear prediction model is an order-1 regressor which can be expressed as \( ax + \beta \), where \( a \) means coefficient of batch latency, and \( \beta \) means a bias value of the model. Details of RandomForest and DNN implementations are presented in Section III-B. Using the three different models, the median ensemble approach of PROFET takes the three predicted values and determines the median value as the predicted latency.

As shown in the figure, PROFET shows the lowest error in MAPE which is a 12.8%. Comparing to the MAPE of DNN, the lowest MAPE among single models, the PROFET shows a 2.4% improvement. But in the case of RMSE, PROFET shows a 24.56% better prediction result than a single DNN model. Using three distinct models with different complexity in an ensemble manner compensates for the weakness of each model, and it greatly improves the generality of the prediction model. For instance, the MAPE and RMSE of Linear model show drastic difference. Careful investigation reveals that the Linear model shows poor prediction accuracy for small models which can make MAPE worse even for small error values.

B. Comparison to the State-of-the-Art

Most contemporary CNN model training latency prediction algorithms adopt the white-box approach while referencing the internal model architecture and hardware configurations as features. We qualitatively argue that the black-box approach of the PROFET is more appropriate in a public cloud environment.
To quantitatively present the prediction accuracy of PROFET with the most recent related work, we compare its performance with Paleo [12], MLPredict [14], and Habitat [11].

First, reproducing the Paleo experiment result necessitates the creation of a training environment that is identical to the one used by the Paleo authors. We discovered that building an identical environment is impractical due to incompatible TensorFlow, GPU driver and library versions. Applying Paleo’s prediction algorithm on a contemporary development environment does not result in the as accurate result as one presented in the original paper, and we decided to compare the result presented in the original publication [12]. Table II compare the prediction accuracy of PROFET and Paleo for common models which are AlexNet [8] and VGG-16 [7]. As shown in the table, PROFET outperforms Paleo. For MAPE, PROFET is 67.85% better and 45.78% better for RMSE.

MLPredict [14] presented an algorithm to predict training time of DNN models across different GPU devices for diverse batch sizes and showed better performance. To compare the prediction accuracy of PROFET with MLPredict, we implemented the MLPredict algorithm following the paper. Table III shows the MAPE and RMSE of MLPredict and PROFET with different batch sizes with VGG16 which showed the best performance in [14]. As shown in the table, the prediction accuracy of PROFET outperforms MLPredict for both MAPE and RMSE. We could observe that the accuracy of MLPredict is poorer than the result presented in the paper [14]. In the original MLPredict paper, the authors mainly predicted training latency of small batch sizes, in the range of 1 to 16. The error rate of MLPredict increases as the batch size becomes larger, and we could conclude the MLPredict algorithm is rather optimized for small batch sizes. However, such small batch sizes are impractical in real-world model training because they increase training time. Recent GPUs available device memory can accommodate larger batch sizes for many well-known DNN implementations, and we believe it is more important to accurately predict the training time of larger batch sizes. In summary, PROFET improves the RMSE by 81.54% comparing to MLPredict.

Last, we compare PROFET with the most recent related work, Habitat [11], which uses detailed profiling results to build a model and predict the training latency across different GPUs. Habitat does not support prediction for varying batch sizes, and we measured the prediction accuracy with fixed batch sizes of 16, 32, and 64 both for PROFET and Habitat. Habitat’s system implementation is open-sourced, and we could reproduce the experimental results in the publication. Table IV presents the prediction accuracy of PROFET and Habitat. For both systems, we use two GPUs of Nvidia T4 and V100. The row with T4 → V100 indicates that the anchor instance is T4, and the target instance to predict the training latency is V100. For different anchors and target GPUs, we predict the training latency of Resnet50, InceptionV3, and VGG16 models and present the average MAPE. We select the GPU device and CNN models that are common to the PROFET and Habitat experiments. On average MAPE, PROFET shows 32.26% lower than that of Habitat.

In summary, PROFET presents the best prediction accuracy among contemporary related work by lowering the MAPE by 32% (Habitat), 68% (Paleo), and 82% (MLPredict). We claim that the higher prediction accuracy of PROFET while using less detailed information than the white-box approach is due to the
TABLE III: The prediction accuracy (MAPE) of VGG16 model for diverse GPU instances of MLPredict and PROFET

| BS  | MLPredict | PROFET | MLPredict | PROFET |
|-----|-----------|--------|-----------|--------|
| 16  | 15.68     | 2.96   | 90.82     | 8.78   |
| 32  | 17.89     | 3.25   | 135.63    | 17.55  |
| 64  | 24.27     | 4.45   | 408.81    | 70.51  |

TABLE IV: The prediction accuracy (MAPE) of Habitat and PROFET with different combination of anchor-target GPUs

|         | Habitat | PROFET |
|---------|---------|--------|
| T4 → V100 | 12.16   | 4.07   |
| V100 → T4  | 7.99    | 8.15   |

V. Related Work

Performance Modeling on Various Hardware: Paleo [12], MLPredict [14] and NeuralPower [13] proposed performance model for DNN training job. To accurately predict the training time or power usage, they use the internal model architecture and GPU specification as input feature of prediction model. Most recent work, Habitat [11] uses a profiling result as PROFET does. However, Habitat uses detailed profiling output which might be inappropriate in a public cloud environment. The aforementioned works refer to the internal architecture of target models (white-box), and algorithm developers may be hesitant to share the architecture.

Using Profiler for DNN Performance Analysis: TBD [34] and Yeung et al. [35] examined the profiling results of various DNN models, hardware, and frameworks and presented an analysis of processing throughput, utilization, and memory consumption of each workload. PerfNetV2 [36] and Ceer [27] uses TensorFlow profiler to collect detailed DNN operation data to build a performance model, and it is similar to PROFET. But it is hard to compare performance with Ceer due to the lack of publicly available artifacts.

DNN System with Performance Modeling: Optimus [37], Cynthia [38], RubberBand [39], and Chaudhary et. al. [40] proposed a DNN tuning and training system that maximizes DNN cluster utilization to reduce processing time. They use the performance model to allocate DNN workload efficiently, or reduce the scale of the GPU cluster. Accurate performance estimation is critical for completing the aforementioned work, and PROFET is complementary to the work in that it can provide the accurate training time of various CNN implementations.

VI. Discussions

Modeling train latency on diverse development platform: During PROFET evaluation, we discovered that applying previous work algorithms on the most recent development platforms did not reproduce the prediction accuracy shown in the original papers, so we had to create a development environment using specific older versions. Most of previous work mentioned the creative definition of input features from the anchor instance and target latency from the instance type that PROFET needs to predict.

VII. Conclusion

Training a CNN model with a GPU device has become the norm because it requires significant computing power, and public cloud service vendors offer a variety of GPU devices elastically. Due to the dynamically changing performance of arbitrary CNN implementations on various GPU devices, it is difficult for an algorithm developer to create an optimal training environment. This paper presented PROFET, which can predict the training latency of arbitrary CNN implementations with diverse configurations on various GPU devices, to aid in the development of an efficient CNN training environment. Without revealing implementation details of CNN implementation, PROFET can predict training latency on multiple distinct GPU devices. Thorough experiments reveal PROFET outperforms contemporary related work. Other than the quantitative superiority of PROFET, predicting latency without revealing implementation detail makes PROFET suitable in a cloud where an algorithm developer and resource providers differ.

Acknowledgment

This work is supported by the National Research Foundation (NRF) Grant funded by the Korean Government (MSIP) (No. NRF-2022R1A5A7000765 and NRF-2020R1A2C1102544).
AWS Cloud Credits for Research program, and the SW Star Lab (RS-2022-00144309) of IITP.

References

[1] JÁGregg Schmidhuber. Deep learning in neural networks: An overview. Neural Networks, 61:85 – 117, 2015.

[2] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. NIPS 2019.

[3] Martin Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. Tensorflow: a system for large-scale machine learning. In USENIX OSDM 2016.

[4] Theano Development Team. Theano: A Python framework for fast computation of mathematical expressions. arXiv 2016.

[5] Norman P. Jouppi, Cliff Young, Nishant Patil, David Patterson, et al. In- system evaluation of datacenter fpga accelerators. In HotChips 2015.

[6] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In CVPR 2012.

[7] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition, 2015.

[8] Geoffrey Y. X. Yu, Yubo Gao, Pavel Golikov, and Gennady Pekhimenko. Habitat: A Runtime-Based computational performance predictor for deep neural network training. In USENIX ATC 2021.

[9] Hang Qi, Evan R. Sparks, and Ameet Talwalkar. Paleo: A performance model for deep neural networks. In ICLR 2017.

[10] Christian Szegedy, Vincent Vanhoucke, and Andrew Zisserman. Return of the devil in the details: Delving deep into convolutional nets. In British Machine Vision Conference 2014, 2014.

[11] Fenghua Peng, Yixin Bao, Yangrui Chen, Chuan Wu, and Chuanxiong Guo. Optimus: an efficient dynamic scheduler for deep learning clusters. In Proceedings of the Thirteenth EuroSys Conference, pages 1–18, 2014.

[12] Zhigang Yang, Shunli Zhang, Zhihong Wang, and Shih-Hao Hung. Toward accurate platform-aware performance modeling for deep neural networks. arXiv 2020.

[13] Tianqi Chen, Mu Li, Yutian Li, Min Lin, Naiyan Wang, Minjie Wang, Tianjun Xiao, Bing Xu, Chiyuan Zhang, and Zheng Zhang. Xgboost: A flexible and efficient machine learning library for heterogeneous distributed systems. arXiv 2015.

[14] U. U. Hafeez and A. Gandhi. Empirical analysis and modeling of compute times of cnn operations on aws cloud. In IISWC 2020.

[15] Flames: Bagging predictors. Machine Learning, 24(2):123–140, Aug 1996.

[16] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2017.

[17] Jingoo Han, M. Mustafa Rafique, Luna Xu, Ali R. Butt, Seung-Hwan Lim, Ziqian Pei, Chensheng Li, Xiaowei Qin, Xiaohui Chen, and Guo Wei. Balancing efficiency and fairness in heterogeneous GPU clusters for DNN training workloads. In USENIX ATC 2021.

[18] Jason K. McKenna, Raul J. Perez, Erica A. Brewer, and Sudharshan S. Vazhkudai. Marble: A multi-gpu aware job scheduler for deep learning on hpc systems. In USENIX HotCloud 2020.

[19] Yuxuan Zhong, Yangluo Gao, Zhihua Zhang, and Shih-Hao Hung. Iteration time prediction for CNN in multi-gpu platforms: modeling and analysis. ACM, 2020.

[20] Guanghui Peng, Yixin Bao, Yangrui Chen, Chuan Wu, and Chuanxiong Guo. Optimus: an efficient dynamic scheduler for deep learning clusters. In Proceedings of the Thirteenth EuroSys Conference, pages 327–342, 2021.

[21] Shubham Chaudhary, Ramachandran Ramjee, Muthian Sivathanu, Nipun Kundra, and Srinidhi Viswanatha. Balancing efficiency and fairness in heterogeneous GPU clusters for deep learning. In Angelos Bilas, Kostas Magouis, Evangelos P. Markatos, Dejan Kostic, and Margo I. Seltzer, editors, EuroSys ’20: Fifteenth EuroSys Conference 2020, Heraklion, Greece, April 27-30, 2020, pages 1:1–1:16. ACM, 2020.

[22] Ziqian Pei, Chensheng Li, Xiaohui Chen, and Guo Wei. Iteration time prediction for CNN in multi-GPU platform: modeling and analysis. IEEE Access, 7:64758–64779, 2019.

[23] Ziqian Pei, Chensheng Li, Xiaohui Chen, and Guo Wei. Iteration time prediction for CNN in multi-GPU platform: modeling and analysis. IEEE Access, 7:64758–64779, 2019.

[24] Ziqian Pei, Chensheng Li, Xiaohui Chen, and Guo Wei. Iteration time prediction for CNN in multi-GPU platform: modeling and analysis. IEEE Access, 7:64758–64779, 2019.

[25] Ziqian Pei, Chensheng Li, Xiaohui Chen, and Guo Wei. Iteration time prediction for CNN in multi-GPU platform: modeling and analysis. IEEE Access, 7:64758–64779, 2019.

[26] Ziqian Pei, Chensheng Li, Xiaohui Chen, and Guo Wei. Iteration time prediction for CNN in multi-GPU platform: modeling and analysis. IEEE Access, 7:64758–64779, 2019.

[27] Ziqian Pei, Chensheng Li, Xiaohui Chen, and Guo Wei. Iteration time prediction for CNN in multi-GPU platform: modeling and analysis. IEEE Access, 7:64758–64779, 2019.

[28] Ziqian Pei, Chensheng Li, Xiaohui Chen, and Guo Wei. Iteration time prediction for CNN in multi-GPU platform: modeling and analysis. IEEE Access, 7:64758–64779, 2019.

[29] Ziqian Pei, Chensheng Li, Xiaohui Chen, and Guo Wei. Iteration time prediction for CNN in multi-GPU platform: modeling and analysis. IEEE Access, 7:64758–64779, 2019.

[30] Ziqian Pei, Chensheng Li, Xiaohui Chen, and Guo Wei. Iteration time prediction for CNN in multi-GPU platform: modeling and analysis. IEEE Access, 7:64758–64779, 2019.

[31] Ziqian Pei, Chensheng Li, Xiaohui Chen, and Guo Wei. Iteration time prediction for CNN in multi-GPU platform: modeling and analysis. IEEE Access, 7:64758–64779, 2019.

[32] Ziqian Pei, Chensheng Li, Xiaohui Chen, and Guo Wei. Iteration time prediction for CNN in multi-GPU platform: modeling and analysis. IEEE Access, 7:64758–64779, 2019.

[33] Ziqian Pei, Chensheng Li, Xiaohui Chen, and Guo Wei. Iteration time prediction for CNN in multi-GPU platform: modeling and analysis. IEEE Access, 7:64758–64779, 2019.