PredictDDL: Reusable Workload Performance Prediction for Distributed Deep Learning

Kevin Assogba*, Eduardo Lima*, M. Mustafa Rafique, Minseok Kwon
Department of Computer Science
Rochester Institute of Technology
Rochester, New York 14623, USA
{kta7930, lima, mrafique, jmk}@cs.rit.edu

***Abstract*—Accurately predicting the training time of deep learning (DL) workloads is critical for optimizing the utilization of data centers and allocating the required cluster resources for completing critical model training tasks before a deadline. The state-of-the-art prediction models, e.g., Ernest and Cherrypick, treat DL workloads as black boxes, and require running the given DL job on a fraction of the dataset. Moreover, they require retraining their prediction models every time a change occurs in the given DL workload. This significantly limits the reusability of prediction models across DL workloads with different deep neural network (DNN) architectures. In this paper, we address this challenge and propose a novel approach where the prediction model is trained only once for a particular dataset type, e.g., ImageNet, thus completely avoiding tedious and costly retraining tasks for predicting the training time of new DL workloads. Our proposed approach, called PredictDDL, provides an end-to-end system for predicting the training time of DL models in distributed settings. PredictDDL leverages Graph HyperNetworks, a class of neural networks that takes computational graphs as input and produces vector representations of their DNNs. PredictDDL is the first prediction system that eliminates the need of retraining a performance prediction model for each new DL workload and maximizes the reuse of the prediction model by requiring running a DL workload only once for training the prediction model. Our extensive evaluation using representative workloads shows that PredictDDL achieves up to $9.8\times$ lower average prediction error and $10.3\times$ lower inference time compared to the state-of-the-art system, i.e., Ernest, on multiple DNN architectures.**

***Index Terms*—Deep Neural Networks, Machine Learning, Performance Prediction, Graph HyperNetwork**

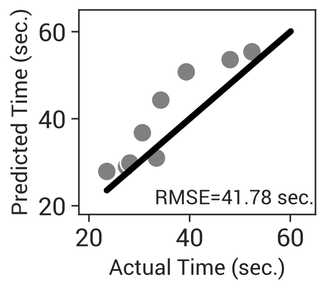
I. Introduction

The performance of training a deep neural network (DNN) in modern data centers is impacted by DNN-specific and hardware-related factors, including the DNN's complexity, and availability and allocation of the compute and storage resources for running the specific deep learning (DL) workload. Performance prediction models approximate the execution time of DL workload, i.e., training a DNN on a dataset using the allocated data center resources, allowing workload managers and schedulers, e.g., SLURM [1], to optimize cluster resource utilization. Modern DL workloads are composed of DNN architectures that change over time [2]–[4] according to business needs, thus requiring a significant overhead of retraining the prediction model to accurately predict the training time. Furthermore, modern DL workloads execute on large-scale distributed computing clusters where the availability of data center resources is highly dynamic [5]. This overhead is dominated by the time to collect execution data from different workload configuration runs to train the prediction model. An ideal prediction model yields a low prediction error, e.g., less than 20% [6], with low overhead after workload changes. However, the search space, comprising multiple neural network (NN) architectures and data center configurations, challenges the applicability of existing performance prediction models that rely on historical execution data to maintain a low error rate. Therefore, reducing the number of times a prediction model is retrained improves its efficiency and enhances its applicability. We address the challenge of accurately predicting the execution time of distributed DL workloads, after a DL workload has changed, and propose a performance prediction model to maintain a low error rate without retraining the prediction model.

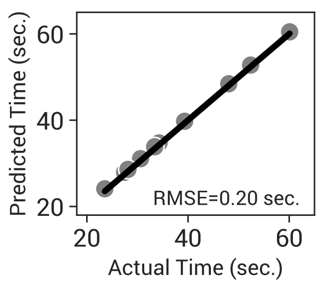
For reducing the retraining overhead, existing prediction models [5]–[8] execute a subset of the target workload, e.g., training with a small amount of data or training using fewer iterations. Depending on the size and execution environment of a DNN, these limited training executions take several hours [5], [9], [10]. As reusing a prediction model without retraining increases the prediction error, the existing performance modeling systems assume that the DL model remains unchanged to maintain high accuracy [6], [7]. However, more DNN architectures are designed for diverse use cases including neural architecture search where performance prediction accelerates the search for the ideal neural network architecture [11].

We predict the training time of DL workloads. We define a DL workload as the training of any DNN model in any computing cluster using any dataset. To accurately estimate the required training time for a DL workload, we inspect the corresponding DNN architecture to generate a vector representation of the computational operations involved in its training process. One major challenge is that predicting the characteristics of a neural network is a computationally prohibitive task akin to the problem of neural architecture search [12]. In general, the design of a neural network follows a trial-and-error approach without a theory that explains the capabilities of a DNN based on its configuration. Moreover, features such as the number of layers and the total number

*Equal contribution

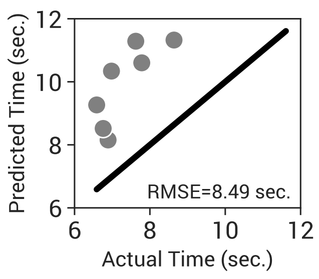


(a) Black Box

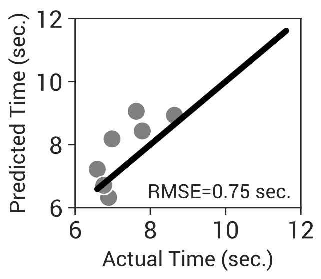


(b) Gray Box

Fig. 1: Comparison of prediction errors using the black box and gray box approaches when predicting the training time of the VGG-16 model.



(a) Black Box



(b) Gray Box

Fig. 2: Comparison of prediction errors using the black box and gray box approaches when predicting the training time of the MobileNet-V3 model.

of learnable parameters are not sufficient to determine the complexity of a neural network [13], [14].

To address the aforementioned challenges, we propose a framework called PredictDDL that predicts the training time of a given DL workload on the available distributed data center resources. We feature the vector representation of the complexity of a DNN architecture and the capabilities of available computing resources to estimate the training time. We formulate the performance prediction as a regression problem with DNN architecture descriptions as part of the input. PredictDDL uses an end-to-end performance prediction engine that leverages Graph HyperNetwork 2 (GHN-2) [15] embeddings to compute the complexity of the given DNN architecture. For prediction, PredictDDL incorporates cluster-specific features, such as the number of servers, the number of floating point operations per second (FLOPS) that can be executed on each server, the available CPU and GPU memory, and CPU and GPU resource utilization. We evaluate the prediction error of PredictDDL using 60 servers (CPU-only and CPU-GPU servers) and 31 DL models. We compare PredictDDL with an existing performance prediction framework, i.e., Ernest [6], and observe that PredictDDL yields 9.8× lower average prediction error than Ernest on six DL workloads. Moreover, PredictDDL improves the inference time by up to 10.3× by reducing the need to retrain DL prediction models.

To the best of our knowledge, this is the first work that uses DNNs' representations obtained from a graph hyper network for performance prediction. Specifically, we make the following contributions:

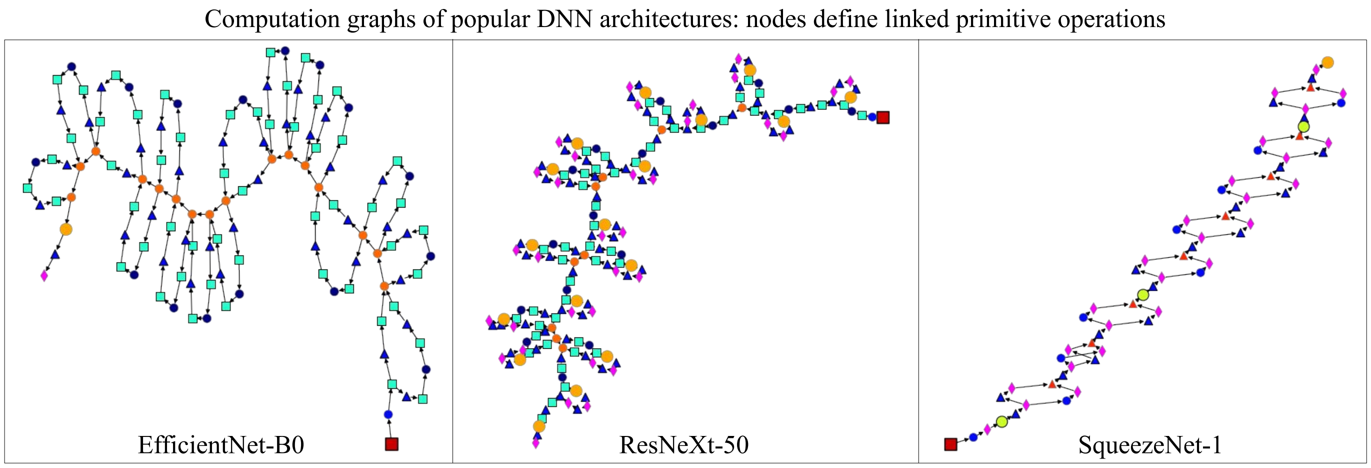
- We propose PredictDDL, a framework that reduces the need for frequent retraining of a performance prediction model for DL workloads in a distributed cluster.
- PredictDDL enables different regression algorithms to be used easily in the prediction model by creating a continuous space that unifies GHN-2 embeddings with cluster description features.
- We conduct a thorough evaluation of PredictDDL and compare it with the state-of-the-art performance prediction models, and show that PredictDDL minimizes the prediction error and inference time.

The rest of the paper is organized as follows. In Section II, we discuss the background and motivation for predicting the performance of DL workloads. In Section III, we present the objectives, overview, architecture, and details of PredictDDL's design. In Section IV, we present the performance evaluation of PredictDDL using diverse DL workloads and compare it with relevant state-of-the-art alternative approaches. In Section V, we summarize existing efforts that are closely related to the work presented in this paper. Finally, we conclude the paper in Section VI.

II. Background and Motivation

A. *Deep Learning Performance Prediction*

Deep learning performance prediction models approximate the performance characteristics, e.g., iteration training time, total training time, the optimal number of epochs, etc., of a DL application to estimate the execution time [16], select the best DNN architecture [11], and ensure fault tolerance [17]. These prediction models either leverage a black box or a gray box approach. We identify a prediction model as a black box that does not leverage the architecture of the DNN, while a gray box approach uses some features specific to the DNN for improving its prediction accuracy. Therefore, black box prediction only considers external characteristics of the DL workload, e.g., data input size and the number of servers in the cluster. The gray box prediction includes features of the DNN architecture, e.g., the number and types of layers, number of parameters, learning rate, and optimization algorithms, in addition to the external features that are used by the black box prediction approaches. As a more fine-grained modeling approach, the gray box approach often requires developers to identify domain-specific features, e.g., the number of filters at each convolution layer, thereby incurring additional complexity in the performance prediction process. They also enlarge the prediction search space, thus negatively impacting the search duration and undermining their usefulness for DL performance prediction. It is inherently challenging to predict the computational requirements of a DNN with a black box approach because several characteristics of the DNN architecture impact the training process [12], [18]. For example,


Computation graphs of popular DNN architectures: nodes define linked primitive operations
EfficientNet-B0
ResNeXt-50
SqueezeNet-1

Marker	🟥	●	🟩	▲	●	●	♦	●
Primitive operation	input	conv	BN	dilated grp conv	sum	avg pool	bias	group conv

Fig. 3: Representation of DNNs as computational graphs to train the GHN-2 model.

DNN architectures become more computationally complex by increasing the dimension of individual layers or stacking up more layers. However, to avoid this additional complexity of enlarged search space in grey box approaches, performance prediction models typically assess the DNN as a black box and do not capture factors essential to accurate predictions. Therefore, it is vital to identify a mechanism to explore the internal features of the DNN for performance prediction without enlarging the search space.

We further motivate the importance of exploring DNN-specific features using the root mean square test error (RMSE) of two DNNs (MobileNet-V3 [19] and VGG-16 [20]). We train each DNN on the CIFAR-10 [21] dataset and collect the training times when varying the number of servers within the training cluster. A detailed description of the servers and their underlying processors (CPUs and GPUs) is presented in Section IV-A. To study the impact of DNN-specific features, we split the collected execution data into training and testing samples following an $80/20$ split ratio. For analysis, we consider the following scenarios: (a) black box approach training a linear regression model with the DNN name, the number of servers, the number of floating point operations per second as features, and the training time as prediction target; (b) gray box approach training a linear regression model with all features in (a) in addition to the number of layers and the number of parameters in each DNN. Figure 1 and Figure 2 report the results. We obtain lower RMSE for all DNNs using the gray box approach. Specifically, we observe up to 99.5% and 91.2% RMSE improvement using a gray box approach on VGG-16 and MobileNet-V3 models, respectively.

These results highlight the benefits of including features that explain the DNN, enabling the linear regression model to capture additional details on the training performance. The black box approach cannot identify the characteristics of the DNN and averages the measurements of the collected training samples resulting in higher prediction errors. Unfortunately, the number of layers and parameters do not comprehensively describe the DNN architecture, leading the existing gray box approaches to explore more features, e.g., kernel size, stride size, and input padding [22], thereby enlarging the search space. Our proposed approach, PredictDDL, addresses this challenge and reduces the search overhead.

B. Graph HyperNetworks: Extracting Vector Representations of DNN Architectures

Graph HyperNetworks (GHNs) [23] are graph neural networks designed to predict neural network parameters. As shown in Figure 3, a DNN architecture can be represented as a directed acyclic graph (DAG) where each node represents a primitive computation operation, e.g., convolution, group convolution, concatenation, summation, averaging, pooling, bias addition, and batch normalization. Edges of this DAG represent the data flow wherein the output of a node is used as the input to the next node. Overall, the DAG of a DNN architecture specifies all operations that are performed on the input data. A GHN takes the computational graph of a DNN as input, as shown in Figure 4, and predicts weights parameters that allow the DNN to achieve faster convergence, lower error, and higher accuracy. GHNs are trained on the same dataset as the target DNN, i.e., they are tailored to the specific task that is performed by the DNN, e.g., image classification on datasets such as CIFAR-10. The output of a GHN is a fixed-size vector that projects the DNN architecture in a continuous n-dimensional architectural space (Figure 5). In PredictDDL, we denote the output vector as DNN embeddings and refer to these embeddings for mapping the DL workloads into points in a search space to determine the most similar DNN architecture. This representation enables our approach

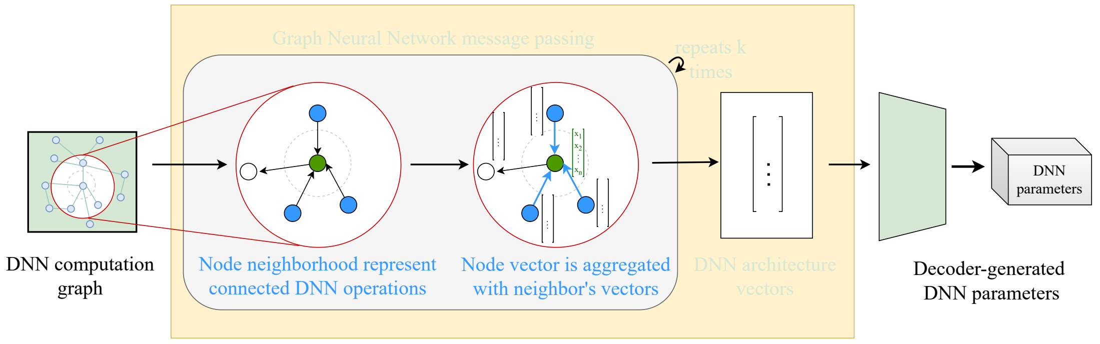

Graph Neural Network message passing
repeats k times
DNN computation graph
Node neighborhood represent connected DNN operations
Node vector is aggregated with neighbor's vectors
DNN architecture vectors
DNN parameters
Decoder-generated DNN parameters

Fig. 4: PredictDDL uses the output of the k-deep graph neural network component of a trained GHN-2 model.

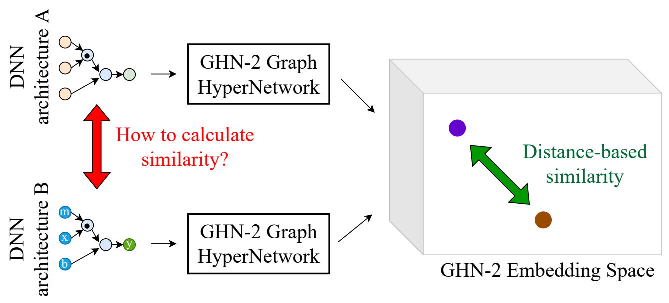


Fig. 5: Distance-based similarity measurement between DNN architectures using fixed-size vector embeddings.

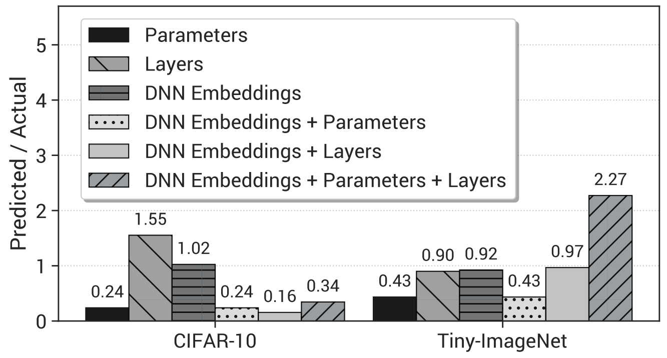


Fig. 6: Impact of DNN architecture features on the accuracy of the DNN's training time prediction. Closer to 1 is better.

to support workloads with multiple DNN architectures without retraining our prediction model.

DNN embeddings generated by GHNs offer a fine-grained representation of the complexity of DNNs and reduce the prediction error of regression models. We evaluate the benefits of the DNN embeddings as compared to other DNN-specific features, such as the number of layers and parameters. For these experiments, we evaluate the error for the CIFAR-10 and Tiny-ImageNet [24] datasets. We use the same experiment setup described in Section IV-A to collect DNN training times and vary the input features of the regression model. We use a second-order polynomial linear regression because it exhibits the best performance in our experiments, as explained in Section IV-B2. Figure 6 summarizes the relative prediction error of each scenario to the actual training time. We observe that GHN yields lower prediction error, e.g., 96.4% and 97.4% lower than features such as the number of layers and trainable parameters, respectively. While the number of layers proves to be more informative than the number of parameters, they perform worse than the GHN-based DNN embeddings that we use in PredictDDL. Note that combining features does not reduce the error value because it introduces duplicate internal representations as the GHN model already considers the types and number of layers and weights parameters.

III. Design of PredictDDL

In this section, we discuss the objectives of PredictDDL as an end-to-end framework for predicting the performance of DL training workloads, present an overview of its design architecture, and describe each of its components in detail.

A. *Design Objectives*

PredictDDL is a reusable performance prediction framework designed to reduce various computational overheads in DL applications, such as neural architecture search, and hyper-parameter optimization. These applications are computationally expensive as they explore tens or hundreds of neural network configurations to identify the best neural architecture or initial weights that satisfy an objective function, e.g., minimize loss, maximize precision, recall, and accuracy. The existing prediction models [6], [7] partially execute a DL workload to collect execution measurements for performance prediction. However, the prediction accuracy degrades when the target DNN architecture is new to the prediction model, i.e., features used by the prediction model are not specific to the target DNN and collected execution measurements do not correspond to the execution environment of the DL workload. Our goal is to predict the training time of any DL workload using a prediction model trained only once on a dataset that does not necessarily contain measurements for the exact DNN used in the DL workload. Specifically, the objectives of PredictDDL are as follows:

- Generate a vector representation of the DNN in the target DL workload to ensure the reusability of the proposed performance prediction model. The vector representation not only identifies the DNN architecture but also embeds the complexity of the DNN model, enabling a seamless association with other DNN models of similar complexity to minimize the prediction error.
- Design a performance prediction framework that can be extended for neural architecture search algorithms [25] or the existing cluster schedulers [26] to optimize the placement of DL training workloads. The performance prediction model should also be extensible for other regression algorithms.
- Minimize the error rate when predicting the completion time of running the training process on a distributed computing cluster of any size.

B. PredictDDL Overview

PredictDDL is a performance prediction and modeling framework that combines DNN-specific features and characteristics of the target computing cluster to predict the execution time of given DL workloads. Our prediction model is driven by the idea of inspecting a DNN architecture by navigating all primitive operations that define the corresponding computational graph. We generate a compact vector representation that characterizes a DNN architecture using the GHN-2 model [15] and uses a regression algorithm that incorporates this representation to predict the training time of a DL workload. The original objective of GHN-2 is to predict weights parameters for DNN training, but intermediate results of the proposed framework capture the DNN's complexity and are beneficial for predicting the training time.

The GHN vector representation converts the original high-dimensional space of DNN architecture computational graphs into a discrete and computationally viable space with a fixed-sized dimension (e.g., 32). It also allows standard regression techniques to associate dependent and independent variables without incurring the overhead of high dimensionality [27]. PredictDDL takes the computational graphs of DNNs as input, converts them into vectors, and uses the distance between a pair of vectors to indicate the similarity of the corresponding DNN architectures. Intuitively, in the vector space, similar DNN architectures are closer than distinct ones, i.e., using cosine similarity, which enables the regression algorithm to find the closest matching DNN architecture for the current workload. This way, we can predict the training time of the target DNN architecture using a regression algorithm trained on experimental measurements of multiple DNN architectures.

Figure 7 shows the performance prediction process of PredictDDL. First, we collect the user's input to PredictDDL, i.e., parameters to describe the DL workload, e.g., size of the input training dataset (1 GB), dataset type (CIFAR-10), tasks (image classification), and the path to the user's training code (step 1). The DL training code captures the DAG representing the DL's computations. Modern DL libraries, e.g., Tensorflow and PyTorch, automatically generate the DAG for the given DL

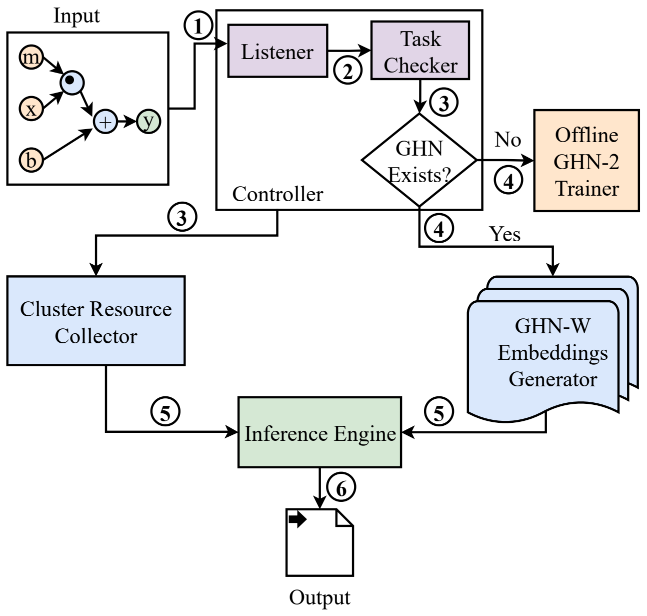


Fig. 7: Workflow of performance prediction in PredictDDL.

model, that we leverage in PredictDDL to make PredictDDL more user-friendly. We use a Listener to receive (step 1) and forward (step 2) the request to the Task Checker for validation (Step 3). If the input dataset does not have a matching pre-trained GHN model, we proceed to an offline training of a new GHN model (Step 4). This scenario requires additional experiments to collect execution samples and retrain the prediction model. However, if the dataset matches a GHN model, irrespective of other parameters in the input request, we generate the vector representation of the target DNN architecture (step 5). Concurrently, we update the information about the characteristics of cluster resources using the Cluster Resource Collector before triggering the prediction of the DNN training time by the Inference Engine (step 6).

C. Inference Engine

The Inference Engine computes the estimated execution time for a dynamically changing target DL workload. It takes as input the data being fed to the workload, a graph describing the DNN architecture, and a cluster configuration that describes the available resources for running the workload. The cluster configuration includes (1) the number of compute servers; (2) associated CPUs; (3) the number of GPUs; (4) available RAM; (5) the number of CPU cores; and (6) the number of FLOPS. We include the number of FLOPS as it encapsulates the processing power of the GPU. Other features, such as the number of streaming multi-processor, and the size of the GPU's high-bandwidth memory, can also be included to capture the characteristics of distributed computing and storage resources that are available for running the given training jobs.

A regression algorithm reads the defined variables and fits a curve while reducing the training and testing errors. Simpler regression algorithms, e.g., linear regression [28], have a lower overhead both at training and inference time; however, they often fail to model the complexity of the prediction [29]. Conversely, overly complex learning algorithms overfit the data and thus perform poorly in a real-world setting for lack of generalization. An ideal regression algorithm needs to adapt to the complexity of DL workloads while maintaining minimum computing overhead.

We train a representative number of regression algorithms, namely linear regression, generalized linear regression with polynomial terms, support vector regression, and multi-layer perceptron, and choose the one that performs best in predicting DL training time in a given cluster. Moreover, PredictDDL also allows users to directly specify their preferred regression model if the model that performs best on the target data is already known. For finding the best-performing regression algorithm, we divide the data into training and test splits and use the test part to estimate the real-world performance.

For describing the computing capabilities of a distributed computing cluster, we make the prediction model agnostic to server configurations. This allows us to process configurations of heterogeneous clusters, i.e., clusters with servers that have different CPU manufacturers, RAM sizes, and storage subsystems. A cluster may have varying states with changes to its available resources at a given time. For example, only 50% of its disk throughput may be available, a fewer number of CPU cores are available than the total installed cores, or there are estimates of unused FLOPS for a CPU core. A cluster under a partial load is modeled by adjusting available capabilities per core, e.g., RAM' is modeled as:

$$RAM' = \frac{RAM}{|cores|} \tag{1}$$

where RAM' represents the estimated *RAM per core*, RAM represents the overall memory available in a server, and $cores$ represents the number of cores present in the CPU. Additionally, we calculate the total available RAM by summing all RAM' over computing cores:

$$AvailableRAM = \sum_{cores} RAM' \tag{2}$$

We apply the same transformation to the disk throughput and the number of FLOPS to calculate available storage throughput and the relative number of FLOPS.

D. Controller

The Controller is the entry point to train GHN models and to predict the training time of a DNN architecture. The controller has a listener to receive and forward incoming requests to the Task Checker for the verification of the requests. In most cases, PredictDDL does not need retraining but uses embeddings generated by the GHN-2 model. However, a new GHN model needs to be trained to generate quality embeddings if the dataset changes as a new dataset with new content (landscape, animal, car images) and type (text, image, or video) introduces new information unseen by the existing GHN models. In contrast, a change in dataset size or adding new samples to the existing dataset does not require retraining. The Task Checker launches the inference procedure directly if a trained GHN model is available for a submitted workload.

E. GHN-based Workload Embeddings Generator

The GHN-based Workload Embeddings Generator selects the closest GHN model out of a set of pre-trained GHN models associated with different datasets. The Embeddings Generator then feeds the GHN model with a computational graph of the target DNN architecture. The GHN model generates vectors that encode the DNN architecture and passes the vectors to the Inference Engine or the Offline GHN Trainer based on the DNN model of the submitted request. Referring to an existing application of GHN models [15], where a correlation study between actual and predicted convergence times, inference speeds, and classification accuracies on clean and corrupt subsets of CIFAR-10 and ImageNet datasets is performed, we identify a similarity between our objective, i.e., predicting the DL training time, and the task of estimating the training convergence time. In [15], we notice an anti-correlation of 0.60 and 0.42 for CIFAR-10 and ImageNet datasets respectively, and therefore hypothesize that GHN models are well-suited to generate vectors for representing DNN architectures.

Our design ensures that PredictDDL finds the closest match based on the cosine similarity in case there is no exact match among DNN architectures in the search space. Consider the submission of a DL workload with a new DNN architecture for predicting its training time. PredictDDL feeds a new computation graph to the GHN-based Workload Embeddings Generator to provide a corresponding vector representation. Formally, GHN takes as input the computational graph of a DNN architecture a, which is a DAG where nodes $V = \{v_i\}_{i=1}^{|V|}$ are operations (e.g., convolutions, fully-connected layers, summations), and their connectivity is denoted by a binary adjacency matrix $A \in \{0,1\}^{|V|\times|V|}$. Nodes are also characterized by a matrix of the initial node features $H_0 = [h_1^0, h_2^0, \dots, h_{|V|}^0]$, where each h_v^0 is a one-hot vector representing the operation performed by the node.

We leverage [23] for GHN definitions and operations. A GHN H_D comprises three modules. The first module transforms the features H_0 of an input node into d-dimensional node features $H_1 \in \mathbf{R}^{|V|\times d}$. This operation is referred to as an embedding layer. The second module reads H_1 along with adjacency matrix A and feeds them into a modified gated graph neural network (GatedGNN) [30]. This GatedGNN mimics the order π in which the operations are performed in the forward (fw) and backward (bw) passes in the computational graph. Intuitively, the forward and backward passes during the evaluation and training phases can be mapped into graph traversal. During the graph traversal, GHN performs iterative message-passing operations to aggregate neighboring nodes' features into message m_v^t, i.e., message received by node v at time t. These messages are continuously aggregated and used to compute the internal embedding vector h_v^{t+1} of node v at time $t+1$ using a recurrent cell function that takes as input

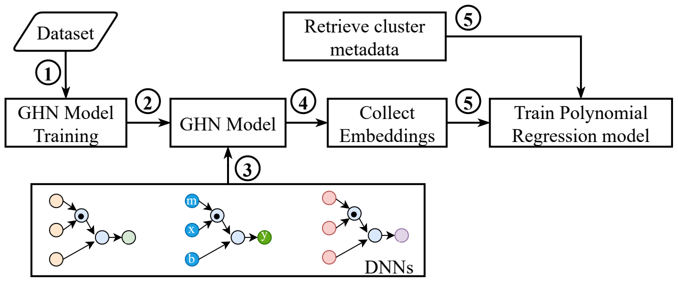


Fig. 8: Workflow of PredictDDL offline training.

the message m_v^t and the current internal embedding vector h_v^t. Each of these inputs is computed as follows:

$$m_v^t = \sum_{u \in \mathcal{N}_v^\pi} MLP(h_u^t), h_v^t = GRU(h_v^t, m_v^t) \quad (3)$$

$$\forall t \in [1, \ldots, T], \forall \pi \in [fw, bw], \forall v \in \pi$$

where T denotes the total number of forward-backward passes; h_v^t corresponds to the features of node v in the t-th graph traversal; $MLP(\cdot)$ is a multi-layer perceptron; and $GRU(\cdot)$ is the update function of the Gated Recurrent Unit [31]. In the forward propagation ($\pi = fw$), $\mathcal{N}_v^\pi$ corresponds to the incoming neighbors of a node defined by A. In the backward propagation ($\pi = bw$), $\mathcal{N}_v^\pi$ corresponds to the outgoing neighbors of the node.

Further enhancements to the original GHN introduced by GHN-2 [15] include: (1) adding virtual edges to the computation graph, as shown in Equation 4; (2) applying operation-dependent normalization to avoid gradient explosion, making the convergence of GHNs more stable; and (3) extending the Differentiable Architecture Search (DARTS) [18] framework to cover a broader set of primitive operations. The authors of GHN-2 generated a synthetic set of 10^6 DNN architectures using DARTS primitives and reported that the GHN model predicts weight parameters better than random initialization. GHN-2 updates the message-passing term m_v^t to incorporate virtual edges calculated by shortest paths between the current node and all other reachable ones:

$$m_v^t = \sum_{u \in \mathcal{N}_v^\pi} MLP(h_u^t) + \sum_{u \in \mathcal{N}_v^{(sp)}} \frac{1}{s_{vu}} MLP_{sp}(h_u^t), \quad (4)$$

$$h_v^t = GRU(h_v^t, m_v^t)$$

$$\forall t \in [1, \ldots, T], \forall \pi \in [fw, bw], \forall v \in \pi$$

where $\mathcal{N}_s^{(sp)}$ are neighbors satisfying $1 < s_{vu} \leq s^{(max)}$, and $s^{(max)}$ is a hyperparameter.

GHN's last module uses the hidden output states h_v^T of the GatedGNN [30] to condition a decoder that produces the parameters w_v^p, e.g., convolutional weights, for each node. Since PredictDDL tries to generate a low-dimensional representation of the DNN architecture, we skip the last module in the original GHN and use the intermediate complexity vector representation as input for our Inference Engine.

TABLE I: Key performance questions that we answer in PredictDDL's evaluation.

Questions	**Sections**
How accurate is PredictDDL at predicting DNN training time?	IV-B1
How do different regression models affect PredictDDL?	IV-B2
How much training data do we need?	IV-B3
Are there any impacts of cluster size on prediction?	IV-B4
Does PredictDDL improve the performance of batch inference?	IV-B5

F. Cluster Resource Collector

The main task of the Cluster Resource Collector is to provide an updated inventory of resources that are available in the cluster. This component leverages a client-server architecture for communication with servers in the cluster. The server module runs on the cluster manager, and all other servers join the cluster through the client module. The Cluster Resource Collector maintains one thread open for new connections to the cluster and launches a pool of threads to collect details about available compute and memory resources in the cluster. Every new server that joins the cluster notifies the Cluster Resource Collector with details about the underlying system and hardware resources, e.g., the number of cores, available memory space, current CPU utilization, and number of FLOPS.

G. Offline GHN-2 Retraining

We retrain the performance prediction model when a new dataset is introduced using an offline retraining procedure, as shown in Figure 8, where the Offline GHN Trainer trains the GHN and prediction models. First, the GHN model is trained using the new dataset. Second, the computational graphs representing DNN architectures are parsed by the trained GHN model to yield fixed-size vectors that compactly store the associated DNN architecture. Concurrently, details on cluster resources are retrieved and used along with the vector representation to train the prediction model. As more cluster configurations are considered, the prediction model will require retraining to learn new features from the performance data collected using the newly added cluster configurations. The GHN-2 model is not associated with the computing environment, and will not require retraining when the same workload is executed on a different cluster.

IV. Performance Evaluation

We evaluate the performance of PredictDDL for predicting the training time of a DL model by measuring (1) the relative error between the predicted and actual training times, i.e., $\frac{Predicted}{Actual}$; and (2) the running time of the PredictDDL prediction engine. We also analyze the impacts of training sample size on prediction error rates to study the overhead of data collection. Through this evaluation, we seek to answer the research questions presented in Table I. Our results show that PredictDDL predicts the training time of DL models with an average relative error of 8%. In comparison to the existing

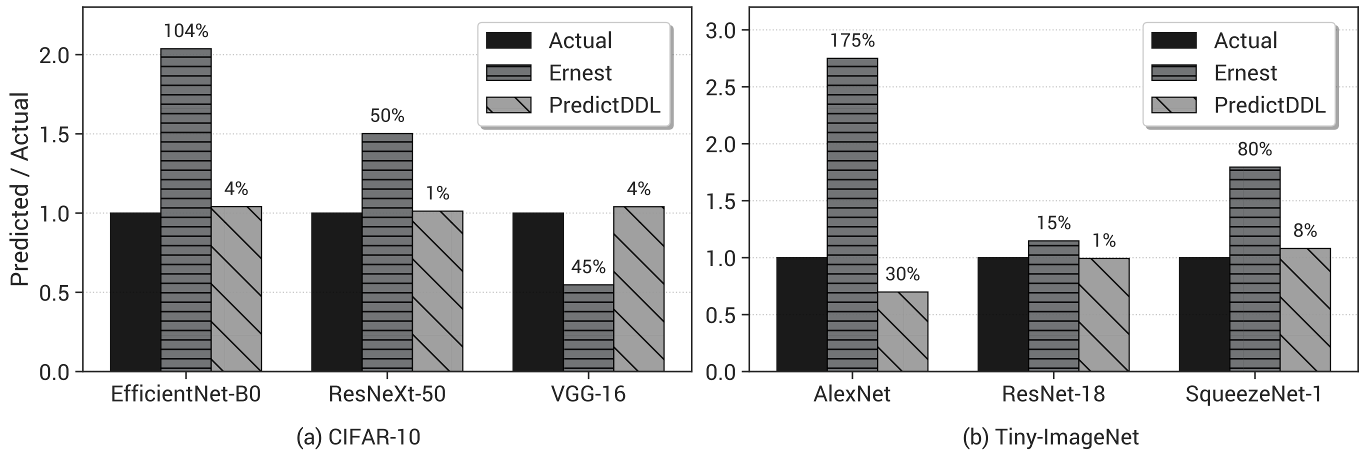


Fig. 9: Prediction error of PredictDDL compared to actual training times and Ernest performance prediction approach. Closer to 1 is better.

state-of-the-art approach, i.e., Ernest, PredictDDL has a 9.8× lower prediction error on average.

A. Experimental Setup

1) Evaluation Testbed: We conduct our experiments using 60 servers from CloudLab [32], [33], specifically 40 CPU-only servers and 20 GPU-based servers. For the CPU-based experiments, we use 20 servers, each of which has two 8-core Intel E5-2630 CPUs and 128 GB memory, and another set of 20 servers, each of which is equipped with one 8-core Intel E5-2650 CPU and 64 GB memory. Each GPU server has two 10-core Intel Xeon Silver 4114 CPUs, 192 GB memory, and one NVIDIA P100 GPU of 12 GB memory connected through PCI Express. Each server has a local disk space of 480 GB and runs Ubuntu 20.04. Moreover, we use CUDA runtime 11.3.1 for GPU-based servers.

2) Deep Neural Network Architectures: We train the prediction engine with 31 image classification DL models from the PyTorch Vision libraries[1]. To diversify our experimental workloads, we select representative models from different architecture families, including ResNet [34], VGG [20], and EfficientNet [35]. We execute the same PyTorch training script for all DL models and only focus on data-parallel training using the PyTorch Distributed Data-Parallel (DDP) module. For GPU-based training, we train each model on dedicated GPUs. Therefore, the study of any interference overhead due to GPU sharing is beyond the scope of this paper. In total, we collect $2,000$ data points by training each DL model by using 1–20 high-end servers.

3) Training Datasets: For each model, we train the studied models using the two representative datasets CIFAR-10 [21] and Tiny-ImageNet [24]. CIFAR-10 (≈163 MB) is an image dataset that contains $60,000$ images grouped into ten classes. Tiny-ImageNet (≈250 MB) is a subset of the large ImageNet dataset and contains $100,000$ images of 200 classes. All the datasets are stored in an external storage device and accessed by the training nodes via the Network File System (NFS). While the dataset is exposed through NFS, we expect a similar trend in the prediction performance for other file systems when the prediction model is trained with data collected from the same cluster. In Table II, we summarize the eight image classification workloads that we use to measure prediction errors in our evaluation.

TABLE II: Summary of training datasets and DL workloads used in evaluating PredictDDL.

Training Datasets	**DL Models**
CIFAR-10	EfficientNet-B0; ResNeXt-50 VGG-16; AlexNet ResNet-18; DenseNet-161 MobileNet-V3; SqueezeNet-1
Tiny-ImageNet	AlexNet ResNet-18 SqueezeNet-1

4) Performance Baselines: We use three performance baselines in our evaluation: (1) the actual training time; (2) Ernest [6]; and (3) PredictDDL. For the actual training time, we execute the studied models on the target cluster configuration and measure its training time. Ernest is a black box approach that predicts training time using linear regression based on historic measurements from the actual training workload's execution. Finally, PredictDDL represents our proposed approach as described in Section III.

B. Performance Results

1) Analysis of Training Time Prediction Error: To evaluate the effectiveness of PredictDDL, we first analyze the relative error of PredictDDL for predicting the training time of DL workloads. We train our prediction engine and Ernest's regression model using a train-test split ratio of $80/20$. Figure 9 shows the prediction errors. We observe that the predictions

[1]https://pytorch.org/vision/0.8/models.html

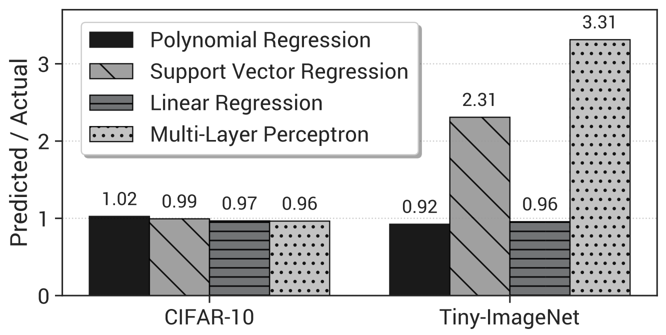


Fig. 10: Impact of different regression models on the prediction accuracy. Closer to 1 is better.

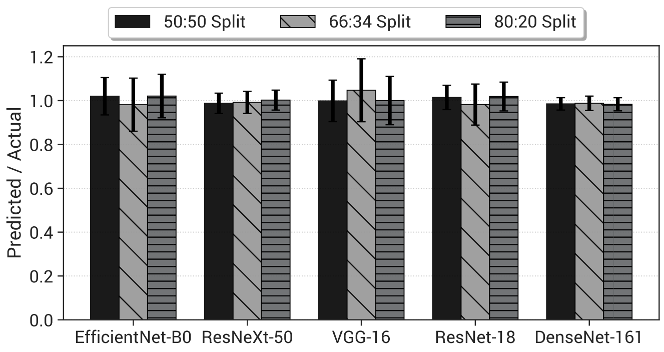


Fig. 11: Impact of the amount of data used to train PredictDDL's predictor. All reported DNNs are trained on the CIFAR-10 dataset. Closer to 1 is better.

of PredictDDL are closest to the actual training times with a 1%–4% prediction error for DNNs trained on CIFAR-10 and a 1%–30% for DNNs trained on Tiny-ImageNet datasets. For all the test workloads, PredictDDL yields a mean relative error of 8%, and on average, PredictDDL reduces a prediction error by $9.8\times$ compared to Ernest. PredictDDL demonstrates its efficiency at approximating the training time due to two main reasons: First, PredictDDL introduces a vector representation of the DNN with the target test workload as an input feature, whereas Ernest evaluates the test workloads merely as black boxes without considering the DNN that is a part of the DL workload. Second, PredictDDL leverages a polynomial regression model, but Ernest uses a simple linear regression model. Note that linear regression predicts well on VGG-16 and ResNet-18, but performs poorly on EfficientNet-B0 and AlexNet. This discrepancy presents a research question for identifying an ideal regression model for targeted DL workloads. We address this question in Section IV-B2.

2) Impact of Different Regression Models on PredictDDL: We compare four regression models to find the approach that yields the lowest prediction error. Specifically, the models are second-order polynomial regression (PR), supporting vector regression (SVR), multi-layer perception (MLP), and generalized linear regression (LR). These models are tuned to identify regression parameters that ensure maximum accuracy. For example, we perform a grid search for SVR considering radial and linear kernels with a trade-off parameter C from 1 to 10^3, an influence indicator γ from 0.05 to 0.5, and ϵ value ranging from 0.05 to 0.2. For MLP, we use a single hidden layer with 1 to 5 neurons. Moreover, we limit the number of neurons to avoid over-fitting. The train and test split ratio remains 80/20 for all the regression models. Figure 10 shows the impact of using different regression models on prediction accuracy. We observe that PR and LR models produce high accuracy for both CIFAR-10 and Tiny-ImageNet datasets. We also observe that SVR and MLP yield low error rates on CIFAR-10, but not on the Tiny-ImageNet dataset. This performance gap can be explained by the configuration differences between different test workloads. For example, DNNs trained on CIFAR-10 leverage GPUs, thus reducing the training duration and exposing smaller values as input for the regression analysis. While PR and LR models achieve similar performance, we identify PR as an ideal regressor compared to other regression models used in our evaluation because of the added benefit of including both the first and second powers of feature values.

3) Sensitivity of the Prediction Error to the Prediction Model's Training Dataset Size: We also study the impacts of different train-test split ratios on the performance of PredictDDL. We split the collected dataset following 50/50, 67/33, and 80/20 ratios to train and test the prediction engine of PredictDDL, where the 80/20 ratio uses 80% of the data for training and 20% for testing. Figure 11 shows the prediction accuracy of the trained regression model on five evaluation workloads. We observe that PredictDDL performs well on all three split ratios, but does not improve in accuracy when the size of the train split increases. Due to the randomness in dividing the train-test samples, the training data does not always contain training samples that are used by the regression model to improve its accuracy for unseen workloads. Therefore, it is essential to consider the relevance of training samples when predicting the training time of unseen workloads. We note that the study and careful selection of training samples is beyond the scope of this paper.

4) Impact of the Training Cluster Size on Performance Prediction Error: We analyze the relative prediction error of PredictDDL on different cluster sizes. In this experiment, we predict the training time of the DL models used in our evaluation when executed on 4, 8, and 16 servers, and report the prediction error. Figure 12 shows that PredictDDL estimates the training time within a minimum of 0.1% and up to 23.5% of the actual time across all workloads used in our evaluation. These results indicate that PredictDDL remains effective irrespective of the scale of the execution environment used for training the given workload.

5) Scalability Analysis of PredictDDL on Batch Performance Prediction Jobs: We also evaluate the overhead of including DNNs' internal representations into the features used by the prediction engine of PredictDDL compared to considering the DNN as a black box. We define the submission

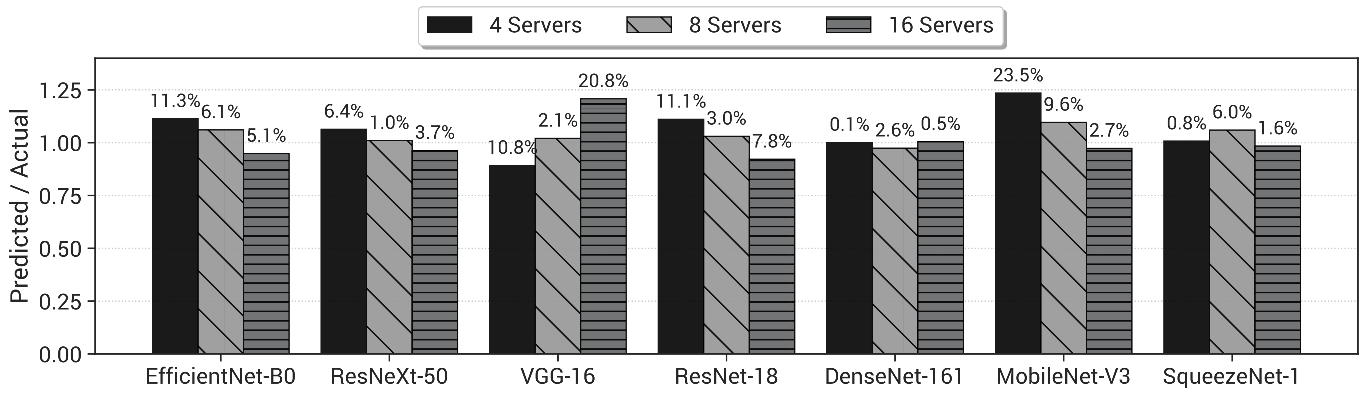


Fig. 12: Impact of cluster size on training DL workloads on the prediction error of PredictDDL. Closer to 1 is better.

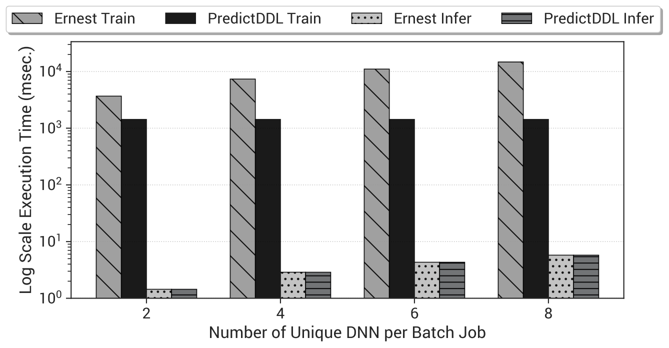


Fig. 13: Prediction model training and execution durations for batches of different DL workloads.

of two or more test workloads from Table II as one batch job and simulate the execution of batch jobs with 2, 4, 6, and 8 DL models. Figure 13 presents the training and inference execution times on a log scale. PredictDDL trains its prediction model only once and can complete all the inference workloads with little changes in the accuracy. In contrast, Ernest needs to retrain its prediction model with new data every time the workload changes to maintain high accuracy. As a result, PredictDDL reduces the total execution time, including training and inference execution times, by $2.6\times$, $5.1\times$, $7.7\times$, and $10.3\times$ for batch jobs of 2, 4, 6, and 8 DL models, respectively. We note that PredictDDL incurs an additional overhead for generating embeddings for the submitted DL models, however, this overhead is amortized as the number of architectures grows, highlighting the scalability benefits of PredictDDL.

V. Related Work

We summarize closely related work including black box performance prediction, analytical performance modeling and search space optimization.

A. Black Box Performance Prediction

Ernest [6] predicts the performance of analytics jobs with high accuracy but does not capture the complexity and dynamics of modern deep neural network architectures. Another popular black box approach, i.e., CherryPick [7], identifies the best cloud configurations for big data analytics workloads using non-parametric Bayesian optimization with a smaller search cost than Ernest, however similar to Ernest, CherryPick is sensitive to workload changes, and requires retraining the prediction model for better cost estimation. Other research works have introduced black box performance prediction models as part of cluster schedulers [8]. For example, Optimus [8] proposes a scheduler that models the training convergence to find the optimal distribution of machine learning workloads. A major limitation of the black box prediction models is that a change in the target workload requires retraining the target DNN to collect the training time through multiple training runs that consume additional memory and computation time. Some prediction models succeeded in decreasing the number of training runs [6], but still are unable to fully adapt to workloads with deep neural network architectures.

B. Analytical Performance Modeling

Several analytical performance models have been introduced to explore the complexity of deep neural networks [36], [37]. Paleo [38] separated training time into computation and communication times to capture the complexity of deep neural network architectures based on their input size, number of FLOPS, parallelization strategies, and bandwidth. Justus et al. [22] conducted an in-depth analysis of the features that are unique to each type of layer, e.g., pooling, recurrent, and convolution. They employed random sampling to reduce the search space. Habitat [39] targets the challenges of selecting cost-efficient GPUs for running deep learning jobs and leverages pre-trained multi-layer perceptrons that take layer dimensions, memory capacity, bandwidth of GPUs, peak FLOPS, and the number of streaming multiprocessors as input. These analytical models predict the performance based on measurements from sample execution of the given DL workload, i.e., for one epoch or a few iterations. However, they either capture a few internal characteristics of the deep neural network or require fine-grained input parameters for accurately predicting the training time of DL workloads.

C. Search Space Optimization

Performance prediction models are extensively used in neural architecture search, hyperparameter optimization, etc., to optimize the search for solutions that satisfy an objective function, e.g., maximum training accuracy, and minimum testing error [40]–[42]. Naive search approaches are expensive as they explore all possible combinations of hyperparameters. Prediction models reduce the number of evaluated combinations [40]. However, the number of parameters to consider, e.g., optimization algorithm, learning rate, batch size, and activation function, remains considerable. PredictDDL uses GHN, a class of GNN [15] that significantly reduces the dimensionality of the search space. The resulting low-dimension representation is also beneficial as more features of the underlying compute cluster are added to the search space because of their importance for accurately modeling the execution time of DL workloads [43].

Existing black box approaches fall short of modeling the complexity of DL workloads with multiple and varying DNN architectures, while current analytical performance modeling approaches fail to capture critical characteristics of the deep neural network or are complex in their design and requirement for fine-grained details on the deep neural network. PredictDDL addresses the limitations of these two classes of performance prediction models by proposing a reusable approach that is more accurate without requiring running the costly retraining process of the prediction model.

VI. Conclusion

We have presented PredictDDL, a framework to predict the training time of distributed deep learning (DL) workloads by transforming the computational graph of their respective deep neural network (DNN) into a vector representation of its complexity using Graph HyperNetworks (GHNs) and incorporating the characteristics of the available resources of the distributed computing cluster for accurately predicting the training time. PredictDDL consists of a performance prediction model that is trained offline and predicts the performance of different DNN architectures without retraining when the workload changes, contrary to the existing black box approaches. We conducted a comprehensive performance evaluation using representative DNNs and training datasets, and observed that PredictDDL outperforms Ernest, a popular state-of-the-art performance prediction model, on all evaluated workloads. PredictDDL reduces the error in predicting the training time of DL workloads by up to $9.8\times$ as compared to Ernest and reduces the total execution time of inference requests by up to $10.3\times$.

Motivated by these results, we plan to investigate the impact of the embedding vector's dimensionality on prediction error, improve PredictDDL's GHN-based embeddings generator to generalize for multiple datasets, study the impact of different training sample sizes and their distributions on the prediction error, and integrate PredictDDL with production-level cluster schedulers as future work.

Acknowledgment

This work is supported in part by the National Science Foundation (NSF) under Awards No. 2106634 and 2106635. Results presented in this paper are obtained using the Chameleon and CloudLab testbeds supported by the NSF.

References

[1] A. B. Yoo, M. A. Jette, and M. Grondona, "Slurm: Simple linux utility for resource management," in *Proceedings of the Workshop on Job Scheduling Strategies for Parallel Processing*. Springer, 2003, pp. 44–60.

[2] A. Verma, L. Pedrosa, M. Korupolu, D. Oppenheimer, E. Tune, and J. Wilkes, "Large-scale cluster management at Google with Borg," in *Proceedings of the 10th European Conference on Computer Systems (EuroSys)*, 2015, pp. 1–17.

[3] C. Lu, W. Chen, K. Ye, and C.-Z. Xu, "Understanding the workload characteristics in Alibaba: A view from directed acyclic graph analysis," in *Proceedings of the 2nd International Conference on High Performance Big Data and Intelligent Systems (HPBD&IS)*, 2020, pp. 1–8.

[4] W.-Y. Chen, K.-J. Ye, C.-Z. Lu, D.-D. Zhou, and C.-Z. Xu, "Interference analysis of co-located container workloads: a perspective from hardware performance counters," *Journal of Computer Science and Technology*, vol. 35, no. 2, pp. 412–417, 2020.

[5] R. Adolf, S. Rama, B. Reagen, G.-Y. Wei, and D. Brooks, "Fathom: Reference workloads for modern deep learning methods," in *Proceedings of the IEEE International Symposium on Workload Characterization (IISWC)*, 2016, pp. 1–10.

[6] S. Venkataraman, Z. Yang, M. Franklin, B. Recht, and I. Stoica, "Ernest: Efficient performance prediction for {Large-Scale} advanced analytics," in *Proceedings of the 13th USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, 2016, pp. 363–378.

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

[8] Y. Peng, Y. Bao, Y. Chen, C. Wu, and C. Guo, "Optimus: an efficient dynamic resource scheduler for deep learning clusters," in *Proceedings of the 13th European Conference on Computer Systems (EuroSys)*, 2018, pp. 1–14.

[9] C. Coleman, D. Narayanan, D. Kang, T. Zhao, J. Zhang, L. Nardi, P. Bailis, K. Olukotun, C. Ré, and M. Zaharia, "Dawnbench: An end-to-end deep learning benchmark and competition," *Training*, vol. 100, no. 101, p. 102, 2017.

[10] M. Commons. (2021) Ml commons training benchmark v1.0. [Online]. Available: https://mlcommons.org/en/training-normal-10/

[11] B. Baker, O. Gupta, R. Raskar, and N. Naik, "Accelerating neural architecture search using performance prediction," *arXiv preprint arXiv:1705.10823*, 2017.

[12] H. Cai, L. Zhu, and S. Han, "Proxylessnas: Direct neural architecture search on target task and hardware," *arXiv preprint arXiv:1812.00332*, 2018.

[13] N. C. Thompson, K. Greenewald, K. Lee, and G. F. Manso, "The computational limits of deep learning," *arXiv preprint arXiv:2007.05558*, 2020.

[14] R. Novak, Y. Bahri, D. A. Abolafia, J. Pennington, and J. Sohl-Dickstein, "Sensitivity and generalization in neural networks: an empirical study," *arXiv preprint arXiv:1802.08760*, 2018.

[15] B. Knyazev, M. Drozdzal, G. W. Taylor, and A. Romero Soriano, "Parameter prediction for unseen deep architectures," *Advances in Neural Information Processing Systems (NeurIPS)*, vol. 34, pp. 29 433–29 448, 2021.

[16] K. Assogba, M. Arif, M. M. Rafique, and D. S. Nikolopoulos, "On realizing efficient deep learning using serverless computing," in *Proceedings of the 22nd IEEE International Symposium on Cluster, Cloud and Internet Computing (CCGrid)*, 2022, pp. 220–229.

[17] M. Arif, K. Assogba, and M. M. Rafique, "Canary: fault-tolerant faas for stateful time-sensitive applications," in *Proceedings of the IEEE/ACM International Conference for High Performance Computing, Networking, Storage and Analysis (SC)*, 2022, pp. 568–583.

[18] H. Liu, K. Simonyan, and Y. Yang, "Darts: Differentiable architecture search," *arXiv preprint arXiv:1806.09055*, 2018.

[19] A. Howard, M. Sandler, G. Chu, L.-C. Chen, B. Chen, M. Tan, W. Wang, Y. Zhu, R. Pang, V. Vasudevan *et al.*, "Searching for mobilenetv3," in *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, 2019, pp. 1314–1324.

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

[21] A. Krizhevsky and G. Hinton, "Learning multiple layers of features from tiny images," *Master's thesis, Department of Computer Science, University of Toronto*, 2009.

[22] 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 (Big Data)*, 2018, pp. 3873–3882.

[23] C. Zhang, M. Ren, and R. Urtasun, "Graph hypernetworks for neural architecture search," in *Proceedings of the 7th International Conference on Learning Representations (ICLR)*, 2019. [Online]. Available: https://openreview.net/forum?id=rkgW0oA9FX

[24] Y. Le and X. Yang, "Tiny imagenet visual recognition challenge," *CS 231N*, vol. 7, no. 7, p. 3, 2015.

[25] T. Elsken, J. H. Metzen, and F. Hutter, "Neural architecture search: A survey," *The Journal of Machine Learning Research*, vol. 20, no. 1, pp. 1997–2017, 2019.

[26] G. Rjoub, J. Bentahar, O. A. Wahab, and A. Bataineh, "Deep smart scheduling: A deep learning approach for automated big data scheduling over the cloud," in *Proceedings of the 7th IEEE International Conference on Future Internet of Things and Cloud (FiCloud)*, 2019, pp. 189–196.

[27] R. Bellman, "Dynamic programming," *Science*, vol. 153, no. 3731, pp. 34–37, 1966.

[28] Y. Li, Y. Sun, and A. Jog, "A regression-based model for end-to-end latency prediction for dnn execution on gpus," in *Proceedings of the IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, 2023, pp. 343–345.

[29] T. Hastie, R. Tibshirani, J. H. Friedman, and J. H. Friedman, *The elements of statistical learning: data mining, inference, and prediction*. Springer, 2009, vol. 2.

[30] Y. Li, D. Tarlow, M. Brockschmidt, and R. S. Zemel, "Gated graph sequence neural networks," in *Proceedings of the 4th International Conference on Learning Representations (ICLR)*, 2016. [Online]. Available: http://arxiv.org/abs/1511.05493

[31] K. Cho, B. Van Merriënboer, D. Bahdanau, and Y. Bengio, "On the properties of neural machine translation: Encoder-decoder approaches," *arXiv preprint arXiv:1409.1259*, 2014.

[32] D. Duplyakin, R. Ricci, A. Maricq, G. Wong, J. Duerig, E. Eide, L. Stoller, M. Hibler, D. Johnson, K. Webb *et al.*, "The design and operation of CloudLab," in *Proceedings of the USENIX Annual Technical Conference (USENIX ATC)*, 2019, pp. 1–14.

[33] R. Ricci, E. Eide, and C. Team, "Introducing cloudlab: Scientific infrastructure for advancing cloud architectures and applications," *; login:: the magazine of USENIX & SAGE*, vol. 39, pp. 36–38, 2014.

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

[35] M. Tan and Q. Le, "Efficientnet: Rethinking model scaling for convolutional neural networks," in *Proceedings of the International Conference on Machine Learning (ICML)*. PMLR, 2019, pp. 6105–6114.

[36] A. Castelló, M. C. Catalán, M. F. Dolz, J. I. Mestre, E. S. Quintana-Ortí, and J. Duato, "Performance modeling for distributed training of convolutional neural networks," in *Proceedings of the 29th Euromicro International Conference on Parallel, Distributed and Network-Based Processing (PDP)*, 2021, pp. 99–108.

[37] Y.-C. Liao, C.-C. Wang, C.-H. Tu, M.-C. Kao, W.-Y. Liang, and S.-H. Hung, "Perfnetrt: Platform-aware performance modeling for optimized deep neural networks," *Proceedings of the International Computer Symposium (ICS)*, pp. 153–158, 2020.

[38] 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. [Online]. Available: https://openreview.net/forum?id=SyVVJ85lg

[39] G. X. Yu, Y. Gao, P. Golikov, and G. Pekhimenko, "Habitat: A runtime-based computational performance predictor for deep neural network training," in *Proceedings of the USENIX Annual Technical Conference (USENIX ATC)*, 2021, pp. 503–521.

[40] C. White, A. Zela, R. Ru, Y. Liu, and F. Hutter, "How powerful are performance predictors in neural architecture search?" *Advances in Neural Information Processing Systems (NeurIPS)*, vol. 34, pp. 28 454–28 469, 2021.

[41] C. White, W. Neiswanger, and Y. Savani, "Bananas: Bayesian optimization with neural architectures for neural architecture search," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 35, no. 12, 2021, pp. 10 293–10 301.

[42] J. Zavatone-Veth and C. Pehlevan, "Exact marginal prior distributions of finite bayesian neural networks," *Advances in Neural Information Processing Systems (NeurIPS)*, vol. 34, pp. 3364–3375, 2021.

[43] B. Huang, M. Boehm, Y. Tian, B. Reinwald, S. Tatikonda, and F. R. Reiss, "Resource elasticity for large-scale machine learning," in *Proceedings of the ACM SIGMOD International Conference on Management of Data*, 2015, pp. 137–152.