IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received 17 November 2023, accepted 27 December 2023, date of publication 10 January 2024,
date of current version 19 January 2024.

Digital Object Identifier 10.1109/ACCESS.2024.3352017

RESEARCH ARTICLE

A Generic Performance Model for Deep Learning in a Distributed Environment

TULASI KAVARAKUNTLA, LIANGXIU HAN, HUW LLOYD, (Member, IEEE), ANNABEL LATHAM, (Senior Member, IEEE), ANTHONY KLEEREKOPER, AND SAMSON B. AKINTOYE

Department of Computing and Mathematics, Manchester Metropolitan University, M15 6BH Manchester, U.K.

Corresponding author: Liangxiu Han (l.han@mmu.ac.uk)

This work was supported in part by the National Overseas Scholarship, India, and in part by the Royal Society-Academy of Medical Sciences, Newton Advanced Fellowship under Grant NAF\R1\180371.

ABSTRACT Performance modelling of a deep learning application is essential to improve and quantify the efficiency of the model framework. However, existing performance models are mostly case-specific, with limited capability for the new deep learning frameworks/applications. In this paper, we propose a generic performance model of an application in a distributed environment with a generic expression of the application execution time that considers the influence of both intrinsic factors/operations (e.g. algorithmic parameters/internal operations) and extrinsic scaling factors (e.g. the number of processors, data chunks and batch size). We formulate it as a global optimisation problem and solve it using regularisation on a cost function and differential evolution algorithm to find the best-fit values of the constants in the generic expression to match the experimentally determined computation time. We have evaluated the proposed model on three deep learning frameworks (i.e., TensorFlow, MXnet, and Pytorch). The experimental results show that the proposed model can provide accurate performance predictions and interpretability. In addition, the proposed work can be applied to any distributed deep neural network without instrumenting the code and provides insight into the factors affecting performance and scalability.

INDEX TERMS Deep learning, analytical modeling, empirical modeling, optimization, differential evolution.

I. INTRODUCTION

Deep neural networks are effective tools for unsupervised data exploration to discover correlation structures. As a result, they are widely used in computer vision, self-driving cars, medical image analysis, video games, and online self-service applications. However, deep neural network architectures such as Googlenet [1], ResNet [2], VGG net [3], and Deep CNN [4] necessitate the use of high computational resources. Training with a large amount of data requires a parallelised and distributed environment, primarily data parallelism, model parallelism, pipeline parallelism, and hybrid parallelism. Performance modelling is essential in quantifying the efficiency of large parallel workloads. Performance models are used to obtain run-time estimates by modelling various aspects of an application on a target system. However, accurate performance modelling is a challenging task. Existing performance models are broadly categorised into two methodologies: analytical modelling and empirical modelling. Analytical modelling uses a transparent approach to convert the model's or applications' internal mechanisms into a mathematical model corresponding to the system's goals, which can significantly expedite the creation of a performance model for the intended system. The existing analytical modelling works investigated deep learning performance modelling and scaling optimisation in distributed environments [5], asynchronous GPU processing based on mini-batch SGD [6], efficient GPU utilisation in deep learning [7], comprehensive analysis and comparison of the performance of deep learning frameworks running on GPUs [8], [9]. However, the major limitation of these works is the poor presentation of the underlying internal operations (i.e., areas of the features' space or specific workload conditions) in the distributed environment.

The associate editor coordinating the review of this manuscript and approving it for publication was Asad Waqar Malik.

Empirical modelling is an excellent alternative to analytical models. In this approach, modelling predicts the outcome of an unknown set of system parameters based on observation and experimentation. It characterises an algorithm's performance across problem instances and parameter configurations based on sample data. Empirical models predict the output of a new configuration on the target machine. Existing works investigated deep convolutional neural networks using asynchronous stochastic gradient descent techniques in a distributed environment [6], Rakshith et al. [10] empirically assessed Horovod's image classification performance, highlighting its advantages over TensorFlow and PyTorch frameworks, Lin et al. [11] presented a GPU cluster-based distributed deep learning performance prediction model. Nevertheless, empirical modelling works cannot provide an unbiased experimental study in a distributed environment using GPUs.

Thus, in this paper, we are inspired by the hybridisation of the analytical and empirical approaches to developing a novel generic performance model that provides a general expression of intrinsic and extrinsic factors of a deep neural network framework in a distributed environment that gives an accurate performance. Specifically, we have developed a generic performance model applicable to any distributed deep neural network without instrumenting the code, which furthermore allows for explaining intrinsic parameters' performance and scalability, providing added value in the field [12]. Our contributions include the following:

- We have developed a generic expression for a performance model considering the influence of intrinsic parameters and extrinsic scaling factors that affect computing time in a distributed environment.
- We have formulated the generic expression as a global optimisation problem using regularisation on a cost function in terms of the unknown constants in the generic expression, which we have solved using differential evolution to find the best fitting values to match experimentally determined computation times. Specifically, we use differential evolution to fit a model for the computing time, which can then be used to inform decisions about the training process rather than optimising the computing time directly.
- We have evaluated the proposed model in three deep learning frameworks, i.e., TensorFlow, MXnet, and Pytorch, to demonstrate its performance efficiency.

The remainder of the paper is organised as follows. Section II discusses related work of the existing performance models in a distributed environment. In section III, we discuss research methodology, i.e., problem description, proposed performance modelling, experiment method and optimisation problem. Section IV discusses an experiment-based evaluation of the effectiveness of our proposals. Finally, section V concludes the paper.

II. RELATED WORKS

This section provides an overview of the existing performance models in a distributed computing environment and differential evolution as a solution to the optimization problem.

A. EXISTING PERFORMANCE MODELS

Performance modelling involves prediction - estimating the performance of a new system, the impact of change on an existing system, or the impact of a change in workload on an existing system [13], [14]. Existing performance modelling of Deep Learning (DL) can be broadly divided into two categories: 1) Analytical modelling and 2) Empirical modelling.

1) ANALYTICAL PERFORMANCE MODELLING OF DL FRAMEWORKS

Yan et al. [5] developed performance modelling to evaluate the impact of partitioning and resourcing decisions on the overall performance and scalability of distributed system architectures' using a DL framework Adam [15]. In addition, performance modelling was also used to develop a scalability optimizer that quickly selects the optimal system configuration to reduce DNN training time. However, the model can only be applied to DL systems, particularly when it has parameter servers and synchronous weights between worker nodes dynamically.

Qi et al. [16] proposed an analytical performance model called Paleo, predicting the deep neural network performance by considering communication schemes, network architecture and parallelization strategies. The results demonstrated that hybrid parallelism performed much better than data parallelism while training the Alexnet model. However, the model did not consider other factors that can affect the overall performance of a model, such as memory usage, data transfer, or communication overhead in distributed environments.

Kim et al. [17] evaluated five popular deep learning frameworks TensorFlow [18], CNTK [19], Theano [20], Caffe-MPI [21] and Torch [22] in terms of their performance on single and multi-GPU contexts. In this work, each framework incorporated and compared different convolution algorithms, such as Winograd, General Matrix Multiplication (GEMM), Fast Fourier Transformation (FFT), and direct convolution algorithms, in terms of layered-wise analysis and execution time. The results have shown that FFT and Winograd algorithms surpass the GEMM and other convolution algorithms. However, the convolution algorithms used by the frameworks provided poor explainability regarding their internal operations.

Shi et al. [9] developed a performance model to evaluate the performance of various distributed deep learning frameworks (TensorFlow, CNTK, MXnet and Caffe) with deep convolutional neural networks (Alexnet, ResNet and GoogleNet models) on the multi-GPU environment. They measured training time, memory usage, and GPU utilization

and compared the frameworks in terms of training time and resource utilization. However, they did not provide a breakdown of the time to divide the minibatch into smaller batches or measure the load imbalance factor, which are critical factors that can significantly affect the training efficiency and performance in a parallel computing environment. Kavarakuntla et al. [23] extended the Shi analytical performance model to evaluate the run-time performance of deep learning frameworks (TensorFlow, MXnet and Chainer) by using the convolutional neural network, multilayer perceptron model and autoencoder model running in the multi-GPU environment. The extended model considered the load imbalance factor and made a layer-wise analysis of a neural network, providing a more comprehensive evaluation of the frameworks' performance. The experimental results showed that the factors influenced the frameworks' performance and considered the load imbalance factor and its importance in a distributed environment.

However, the models mentioned above have poor explainability and were not generic; they were developed for specific architectures, which cannot be applied to a wide range of networks.

2) EMPIRICAL MODELLING OF DL FRAMEWORKS

Empirical modelling builds models through observation and experimentation, which is antithetical to analytical modelling.

Oyama et al. [6] proposed a performance model for predicting the statistics of an asynchronous stochastic gradient descent-based deep learning system, potentially improving the model's performance by optimising the hyperparameters, such as Mini-batch size and gradient staleness. They did not consider parallelisation methods and applied direct weights synchronised among GPUs. The study results showed that the proposed method could predict the statistics of asynchronous SGD parameters, including sweeping dataset time, mini-batch size, staleness, and probability distributions of these essential parameters. However, the work did not address the issue of communication overhead and network latency, which can significantly affect the performance of distributed deep learning systems.

Rakshith et al. [10] presented an empirical study of the performance of Horovod, a distributed deep learning framework, for image classification tasks. They evaluated the performance of Horovod on two popular image datasets, CIFAR-10 and ImageNet, using a cluster of machines with varying numbers of GPUs. It also compared the performance of Horovod to other distributed deep learning frameworks, such as TensorFlow and PyTorch, and found that Horovod achieved better performance in certain scenarios. They provided recommendations for optimising Horovod's performance on large-scale image datasets, such as using efficient data loading and preprocessing techniques and optimising the communication and synchronisation between the machines. However, the experimental configuration utilised in the research might not accurately reflect real-world situations in which the underlying hardware and network setups may differ substantially.

Lin et al. [11] proposed a performance prediction model for distributed deep learning on GPU clusters that considers both the network topology and communication patterns of the trained deep learning model. The model considered the communication and computation times for each layer in a deep neural network and used a prediction model that is more sophisticated than a simple linear regression approach to predict the total training time. The work evaluated the model on several deep learning benchmarks and showed it achieved higher accuracy in predicting training time than existing models. The model can also be used to optimise the performance of distributed deep learning by finding the optimal configuration of GPU nodes and reducing the training time. However, the assessment of the proposed model was confined to three distinct GPU clusters, potentially limiting its generalizability to other GPU clusters or distributed DL architectures.

Most recently, a new approach named hybrid model has been proposed [24] by combining the elements of analytical modelling and empirical modelling for better performance prediction developed in other fields. We are inspired by this idea and proposed a model that gives insights into the intrinsic parameters' performance and scalability of the extrinsic parameters. Unlike the existing works summarised in Table 1, we developed a generic expression applicable to any distributed deep neural network without instrumenting the code and enabling functionality such as explaining internal parameters' performance and scalability.

B. DIFFERENTIAL EVOLUTION

Differential Evolution (DE) was developed by Storn et al. [25] as an algorithm to solve various complex optimisation problems such as motor fault diagnosis [26], structure prediction of materials [27], automatic clustering techniques [28], community detection [29], learning applications [30] and so on. The algorithm finds the best solution by maintaining a population of individual solutions and creating new offspring by combining existing ones according to a specific process. The offspring with the best objective values are kept in the next iteration of the algorithm so that an individual's new objective value is improved, consequently forming part of the population. Otherwise, the new objective value is discarded. The process repeats itself until a specific termination condition is satisfied [31]. It is similar to other evolutionary algorithms such as Genetic Algorithm (GA) [32] by applying mutation, crossover, and selection operators to determine the population toward better solutions. In contrast to the genetic algorithm, the differential evolution algorithm imparts mutation to each individual while transferring them to the next generation. In its mutation procedure, for each solution, three more individuals are picked from the population, and as a consequence, a mutated individual

TABLE 1. Analysis of the existing performance modelling methods.

Papers	Performance modelling methods	Adaptability and Generalisation	Distributed Implementation
Oyama *et al.* [6]	Analytical modelling	Non-generic	Multiple nodes
Qi *et al.* [16]	Analytical modelling	Non-generic	Multiple nodes
Heehoon Kim *et al.* [17]	Analytical modelling	Poor explainability and non-generic	Multi-GPU
Shi *et al.* [9]	Analytical modelling	Poor explainability and non-generic	Multi-GPU
Oyama *et al.* [6]	Empirical modelling	Non-generic	Multi-GPU
Rakshith *et al.* [10]	Empirical modelling	Non-generic	Multi-GPU
Z.Lin *et al.* [11]	Empirical modelling	Non generic	Multi-GPU, but limited to three distinct GPU clusters
Our method	Analytical modelling and Empirical modelling	Generic	Multiple nodes and Multi-GPU

is produced. It is determined based on the fitness value whether or not the first individual selected will be replaced. In differential evolution, the crossover is not the primary operation, as it is in the genetic algorithm. In recent times, several works have been proposed to use DE for neural network optimisation [33], [34], [35].

However, none of the works mentioned above used DE to analyse and evaluate the performance of the classification tasks of neural networks with many processes in a distributed environment with the goal of finding the best-fit values by minimising the regularised cost function.

III. METHODOLOGY

A. PREDICTION MODEL

The evolution of our prediction model was a systematic process aimed at enhancing its accuracy and performance in estimating execution times. Initially, the model was set up with vector constants (a, p, q, and C) assigned random values. Subsequently, we introduced the Differential Evolution (DE) algorithm, which iteratively refined these constants by generating trial vectors through parameter vector combinations and selecting the best-performing ones. regularisation techniques, including L1 and L2 regularisation, were incorporated to prevent overfitting. The model was trained using a comprehensive dataset of input features and corresponding execution times, allowing it to adapt its constants iteratively. The evolution process continued until convergence when the optimized constants were finalized. A separate validation dataset was employed to validate the model, and comparative analyses against baseline methods were conducted to demonstrate its superior predictive accuracy.

B. THE GENERIC PERFORMANCE MODEL

Given an application consisting of a number of processes in a distributed environment, the execution time of the application can be considered from two levels: 1) Execution time of internal processes of the application (for example, intrinsic parameters of the application) and 2) External scaling factors that affect the computing efficiency (such as a number of machines/processors or data chunks or batch size). A generic performance model for computing total computational time(t) per iteration of an application can be described as follows:

$$t(I, E) = t_I(I) f_E(E) + C \tag{1}$$

Here, we represent intrinsic parameters, E represents extrinsic parameters, t_I represents the time affected by intrinsic parameters, f_E represents extrinsic scaling factors affecting computing performance, and C is a constant. Also, we represent internal time as t_I and can be represented as:

$$t_I = \sum_{i=1}^{n} a_i I_i^{p_i} \tag{2}$$

Basically, intrinsic parameters represent model parameters of the deep neural network, as shown in figure 1. We represent the individual processes as a polynomial in terms of the internal parameters. In (2), the coefficients a_i relate to the relative importance of the processes, and the powers p_i relate to the computational complexity. The external factors are related to scaling, and they will appear as multiplicative terms with different powers in the computation of the external scaling factor f_E, which is given by:

$$f_E = \prod_{j=1}^{m} E_j^{q_j} \tag{3}$$

Here, the powers q_j give information about scalability. By substituting the t_I and f_E in (1), the computational time (t) is given as follows:

$$t(I, E, x) = \left(\sum_{i=1}^{n} a_i I_i^{p_i} \right) \prod_{j=1}^{m} E_j^{q_j} + C \tag{4}$$

Here $\boldsymbol{x} = \{a_1, \ldots, a_{n_I}, p_1, \ldots p_{n_I}, q_1, \ldots q_{n_E}, c\} \in \mathbb{R}^M$ is a vector formed by combining $\boldsymbol{a}, \boldsymbol{p}, \boldsymbol{q}$ and coefficient $\boldsymbol{C}$. It encodes the information necessary to describe how the intrinsic and extrinsic parameters influence the computation time. The functional diagram of the proposed performance

model is shown in figure 2. In (4), the intrinsic parameters I and extrinsic parameters E are the known input values. $\boldsymbol{a}, \boldsymbol{p}, \boldsymbol{q}$ and coefficient $\boldsymbol{C}$ are unknown constants. We compute the optimal values of these unknown constants (total: $M = 2n_I + n_E + 1$) using the differential evolution algorithm. Before going to the cost function formulation of differential evolution algorithm [36], consider the representation of the following input samples (i.e., intrinsic and extrinsic parameters). For every internal process and extrinsic parameter, there are too many hyperparameters. So, we have applied random sampling to ensure that every hyperparameter in the population has an equal opportunity of being selected to obtain measured time. Here, the methodology used for measured time is the time taken for an iteration of an epoch. We computed iteration time as the difference between an iteration's end and starting times.

$$\text{Let, Intrinsic parameters: } I_{i,k}, \quad i \in [1, n_I], \quad k \in [1, N] \tag{5}$$

$$\text{Extrinsic parameters: } E_{j,k}, \quad j \in [1, n_E], \quad k \in [1, N] \tag{6}$$

$$\text{measured time-per-iteration: } t_k, \quad k \in [1, N]. \tag{7}$$

Here, i, j denote the input feature indices. $k \in [1, N]$ indicate the sample index in dataset $\boldsymbol{D}$. N is the number of input samples in $\boldsymbol{D}$.

C. GLOBAL OPTIMISATION USING DIFFERENTIAL EVOLUTION

Given the generic expression as shown in (4), as mentioned in an earlier sub-section, we find the best-fit values of $\boldsymbol{a}, \boldsymbol{p}, \boldsymbol{q}$ and C by minimizing a cost function. We formulate the cost function as the mean absolute difference between the predicted execution time and the actual measured times as follows:

$$f(x) = \frac{1}{N} \sum_{k=1}^{N} \left| t_k - \hat{t_k} \right| \tag{8}$$

where N as number of data samples, t_k=measured value, $\hat{t_k}$ = predicted value where $\hat{t_k} = \left(t_{I_k, E_k, x}\right)$.

To solve the above problem, we have used the differential evolution algorithm (DE) and applied DE to the (4). Here $\boldsymbol{a}, \boldsymbol{p}, \boldsymbol{q}$ and C are combined into an m-dimensional vector. We use the implementation of differential evolution from the *scipy* python package, with default values of the hyperparameters. We enforce limits of $(0 \ldots 1000)$ for constants and coefficients (a, C) and $-5 \ldots 5$ for powers (p, q).

D. REGULARISATION

A globally optimized, unconstrained model may be prone to overfitting or producing unstable solutions with high parameter variance. To address these issues, we introduce a regularisation term to the cost function. regularisation achieves the best fit by introducing a penalizing term in the cost function, which assigns a higher penalty to complex curves. So, we are motivated to apply regularisation to our performance model. Generally, regularisation can be defined as:

$$f_{\text{reg}}(x) = f(x) + \lambda . L \tag{9}$$

where λ controls the bias-variance trade-off, and L is some measure of the complexity of the model. There are two types of regularisation techniques: (a) Lasso regression (L1) form. (b) Ridge regression (L2) form. Firstly, L1 regularisation, also called a lasso regression, adds the absolute value of the magnitude of the coefficient as a penalty term to the loss function. The L1 regularisation solution is sparse. Secondly, L2 regularisation, also called ridge regression, adds the squared magnitude of the coefficient as the penalty term to the loss function, and its solution is non-sparse. In L1 regularisation, L1 (Lasso) shrinks the less important features coefficient to zero, thus removing some features altogether. L1 works well for feature selection in case we have a huge number of features. In L2 regularisation, it adds the penalty as model complexity increases. The regularisation parameter λ penalizes all the parameters except intercept so that the model generalizes the data and won't overfit. Ridge regression adds the Squared magnitude of the coefficient as a penalty term to the loss function. We have applied both regularisations to the performance model. Now, the model of both L1 and L2 regularisations is as follows:

$$f(x) = \frac{1}{N} \sum_{k=1}^{N} \left| t_k - \hat{t_k} \right| + \lambda . \sum_{k=1}^{N} |x| \tag{10}$$

$$f(x) = \frac{1}{N} \sum_{k=1}^{N} \left| t_k - \hat{t_k} \right| + \lambda . \sum_{k=1}^{N} x^2 \tag{11}$$

Thus, applying the regularisation term λ reduces the bias-variance trade-off in the internal processes.

IV. EXPERIMENTAL EVALUATION

To evaluate the performance of the proposed model, we have applied our approach to three deep learning frameworks and conducted extensive experiments. The main goal is to investigate how well the predicted execution time fits the experimentally measured time.

A. SYSTEM CONFIGURATION

We implement the experiments on a single node containing three GEFORCE RTX 2080 GPUs, each with 2.60 GHz speed and 16 GB GPU RAM, to study and compare the performance of three popular frameworks: TensorFlow, PyTorch and MxNet. The node also consists of a 2.81GHz speed CPU machine, 25Gbps network bandwidth and a CUDA-10.2 with a Linux operating system. Furthermore, the node consists of various software configurations/installations, including PyTorch 1.2.0, Torchvision 0.4.0, Python 3.6, TensorFlow 2.1.0 and MXnet 1.6.0.

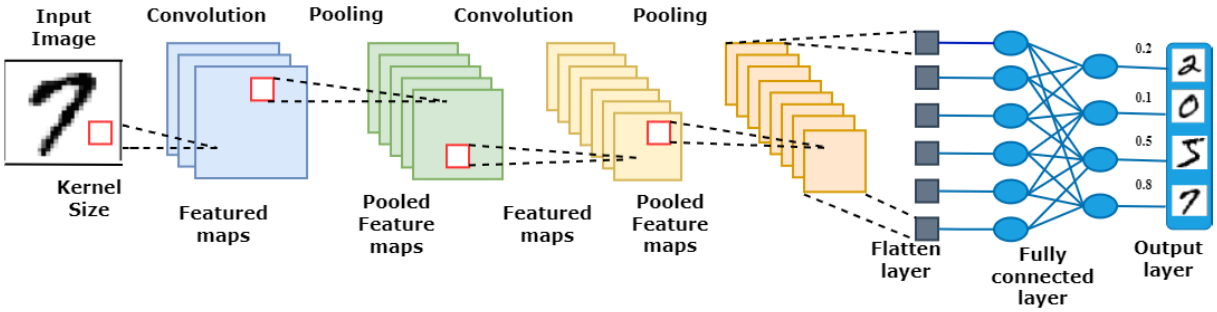


FIGURE 1. Internal processes involved in a convolutional neural network.

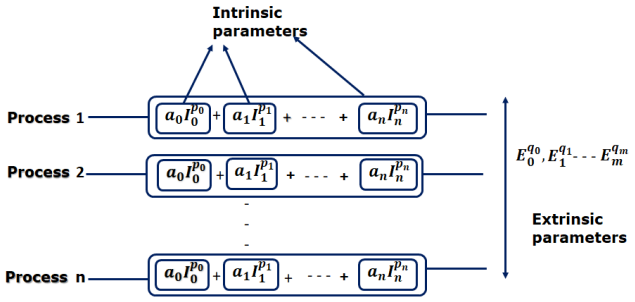


FIGURE 2. Functional diagram of proposed performance model.

B. DATASET AND MODEL SELECTION

We evaluate the proposed performance evaluation approach using a CNN architecture, LeNet-5, which Yann LeCun proposed in 1998 as a neural network structure for handwritten font recognition. It consists of two convolutional layers, two fully connected layers, pooled layers for cross-combination and an output layer that predicts values via the fully connected layer. Besides, LeNet-5 works well with handwritten datasets [37]. It also reduces the number of parameters and can automatically learn features from raw pixels [38].

We train LeNet-5 on three popular datasets, MNIST, fashion-MNIST and CIFAR-10, using three popular deep learning frameworks: TensorFlow, PyTorch and MxNet, in a distributed environment. The distributed environment in our context refers to a multi-GPU training paradigm within a single node rather than an inter-node distributed system. MNIST [39], [40] is a database of handwritten digits derived by the National Institute of Standards and Technology (NIST) for learning techniques and pattern recognition methods with a little effort on pre-processing and formatting. It is a subset of the NIST Special Database 19. Each image represents 28 × 28 pixels. The MNIST database contains 60,000 training and 10,000 testing images, divided into four files: training set images, training set labels, testing set images, and testing set labels. Fashion-MNIST [41] serves as a direct drop-in replacement for the original MNIST dataset for benchmarking machine learning algorithms. Each example is a 28 × 28 grayscale image associated with a label from 10 classes. The CIFAR-10 dataset [42] contains 60,000 images with 32 × 32 pixels. The images are classified into ten classes - aeroplane, automobile, bird, cat, deer, dog, frog, horse, ship, and truck; each has 6,000 images.

C. PERFORMANCE METRICS

The scalability and mean absolute percentage error(MAPE) are selected as performance metrics for run-time evaluation on three different frameworks. Scalability is measured in the powers of external parameters as shown in (3). The MAPE can be defined as:

$$f(x) = \frac{1}{N}\sum_{k=1}^{N}\frac{\left|t_k - \hat{t_k}\right|}{t_k} \tag{12}$$

where t_k = measured value, $\hat{t_k}$ = predicted value, n = total number of data points.

The experimental evaluation aims to evaluate the proposed performance model and find the best-fit values using the differential evolution algorithm; MAPE is used to evaluate the closeness of this fit and the quality of the *performance model*. The scaling parameters are used in our proposed evaluation model to evaluate the performance of the deep learning frameworks.

D. EXPERIMENTS

We have conducted a set of experiments to evaluate the proposed model:

1) Performance Evaluation of Deep Learning Frameworks using the proposed performance model with and without regularisation. Specifically, we have applied the proposed performance model to three deep learning frameworks: TensorFlow, MXnet, and PyTorch, under two circumstances, with and without regularisation.
2) Comparison of the proposed model with the two commonly used black box machine learning models: random forest regressor and support vector regressor. We have also compared our proposed model with two widely used black-box machine learning models and demonstrated its performance and interpretability.

We assess the proposed performance evaluation approach by modelling distributed training of a CNN architecture on a multi-GPU system. In our experiment, we have

TABLE 2. Parameters of the performance model, with ranges of values sampled in the experiments.

Index	Name	Set of possible values considered
Intrinsic parameters		
1	Kernel size	{2,3,4,5}
2	Pooling size	{2,3,4,5}
3	Activation function	{Relu, Tanh, Sigmoid}
4	Optimizer	{Adam, SGD}
5	Image_dataset_name	{MNIST,Fashion-MNIST,CIFAR-10}
6	Number of filters	{4,8,16,32,64}
7	Learning rate	$\{0.1,0.01,0.001,10^{-4}, 10^{-5}, 10^{-6}\}$
8	Padding_mode	{valid, same}
9	Stride	{1,2,3}
10	Dropout probability	{0.2,0.5,0.8}
Extrinsic parameters		
11	Number of GPUs	{1,2,3}
12	Batch size	{8,16,32,64,128}

performed the distributed training of LeNet-5 on MNIST, fashion-MNIST and CIFAR-10 datasets using the three deep learning frameworks. The values of the experimental training parameters are created by applying random sampling on a set of intrinsic and extrinsic parameters and its corresponding average training time taken by a deep CNN architecture per iteration. Table 2 shows intrinsic and extrinsic parameters and their possible values. The intrinsic parameters are the model's hyperparameters, including kernel size, pooling size, activation function, etc. The number of GPUs and the batch size are extrinsic factors since these affect the scaling over multiple processes.

The experiments involve several trials in which we measure the time for a single training iteration using randomly selected intrinsic and extrinsic parameter values. We conduct 1500 trials to prepare a dataset of 1500 data samples. The experimental data for 900 trials are used to fit our performance model or train the standard black box models for comparison. The remaining 600 are used to evaluate the test and validation models. Finally, the experimental parameters are used to build three performance evaluation models, such as the Differential evolution (DE) algorithm with and without using regularisation models and two standard black box models. We run each fit ten times with different random seeds to obtain the mean and standard deviation for each of our fitted parameters. The performance of these models and their corresponding results are explained in the subsequent subsections.

E. RESULTS AND ANALYSIS

This section shows the results of our proposed performance model for three popular deep neural networks, i.e., TensorFlow, MXnet, and PyTorch. We evaluate the performance model with and without regularisation and compare it with standard black box regression models such as support vector regressors and random forest regressor. Tables 3 and 4 compare intrinsic parameters and scalability in various frameworks with and without using regularisation. Table 5 shows mean absolute percentage error values on predictions of the performance models using L1 and L2 regularisation.

1) PERFORMANCE EVALUATION OF DEEP LEARNING FRAMEWORKS USING THE PROPOSED PERFORMANCE MODEL WITHOUT REGULARISATION

We applied a differential evolution algorithm to our proposed model and evaluated it using the three deep learning frameworks. The actual execution time for training the model using the three frameworks is recorded, and predicted execution times are also generated. Figure 3 shows the scatter graph of the predicted execution times from the proposed model plotted against the actual execution time. The linear fit to the straight line determines how well the model can predict unseen configurations. We find best fit constant coefficients for all frameworks are shown in Table 3.

The results show stable and consistent fits for the extrinsic parameters and the additive constant C, indicating that the scalability results are accurate. The higher variances in the intrinsic parameters are reduced by using regularisation. Table 3 shows that the model gives broadly consistent performance for the constant coefficients, representing the relative importance of the process controlled by categorical parameters. For instance, Adam has a large constant for the activation function coefficients and takes more training time than SGD in Pytorch and TensorFlow frameworks, while SGD has the highest training time with the MXnet framework. The *padding* parameter, which is categorical with two possible values *valid* and *same*. *same* shows better performance for the *valid* mode.

2) PERFORMANCE EVALUATION OF DEEP LEARNING FRAMEWORKS USING THE PROPOSED PERFORMANCE MODEL USING REGULARISATION

We have applied the regularisation on a cost function to the proposed performance model to optimise the vector constants and reduce high variance in intrinsic parameters in three deep learning frameworks. We applied both regularisations to our model and compared the results of L1 and L2. The MAPE, MSE, and RMSE results are less in L2, as shown in Table 5. We also considered L2 regularisation appropriate for our performance model and applied various regularisation parameter values in logarithmic scale in L1 and L2 to find better λ parameter. In Figures 7(a) and 7(b), we found that the R2 score deteriorates when the λ value is less than 0.001. For instance, when $\lambda = 0.001$, the model fits well, and the model gives broadly consistent performance for the constant coefficients and represents the relative importance of the process controlled by categorical parameters. Furthermore, in Table 4, we can see that the model gives extensively consistent performance for the constant coefficients, representing the relative importance of the process controlled by categorical parameters. The results show that the performance model using regularisation is a generalised model with optimised good fits in all the frameworks. For example, for *padding* coefficients, *same* parameter takes more training time than *valid* parameter in all frameworks. For activation function

TABLE 3. Derived intrinsic and extrinsic parameters from the differential evolution-optimized performance models for the three deep learning frameworks. Parameters are given as the mean and standard deviation over ten fits. *a* and *p* represent coefficients and powers, respectively, of a term representing an intrinsic parameter, whereas *q* is power in a multiplicative term representing an extrinsic (scaling) parameter.

	Mxnet		Pytorch		TensorFlow	
Intrinsic parameters	a	p	a	p	a	p
Filter size	554.87 ± 311.73	-4.06 ± 0.53	423.36 ± 256.88	-2.88 ± 1.04	346.73 ± 216.24	-3.22 ± 0.78
Kernel size	10.57 ± 7.05	-4.10 ± 0.70	168.54 ± 123.27	-2.34 ± 1.82	54.78 ± 32.91	-4.00 ± 1.41
Pool size	18.08 ± 5.17	-4.21 ± 0.46	209.14 ± 186.87	-3.31 ± 0.92	79.45 ± 53.53	-3.48 ± 1.33
Learning rate	459.50 ± 258.52	3.68 ± 0.62	489.52 ± 221.63	3.21 ± 0.70	458.34 ± 278.03	3.26 ± 0.91
Stride	17.29 ± 6.12	-0.83 ± 0.23	140.64 ± 138.62	-0.63 ± 0.58	29.00 ± 14.54	-1.85 ± 0.90
Dropout probability	1.79 ± 0.75	2.24 ± 1.62	437.06 ± 184.32	1.80 ± 1.66	10.23 ± 9.51	1.87 ± 1.62
Same	2.50 ± 0.97	-	11.02 ± 5.09	-	6.14 ± 1.54	-
Valid	1.56 ± 0.96	-	0.77 ± 1.81	-	1.61 ± 2.24	-
Sigmoid	23.25 ± 10.23	-	475.92 ± 139.65	-	251.57 ± 122.01	-
Relu	21.90 ± 10.40	-	475.56 ± 137.27	-	255.93 ± 122.35	-
Tanh	23.14 ± 10.30	-	444.48 ± 138.25	-	254.28 ± 121.27	-
MNIST	35.75 ± 12.81	-	815.62 ± 69.44	-	232.24 ± 108.77	-
Fashion-MNIST	35.94 ± 12.75	-	815.68 ± 68.39	-	231.33 ± 109.93	-
CIFAR-10	18.57 ± 12.68	-	308.73 ± 53.32	-	124.56 ± 108.01	-
SGD	16.68 ± 10.10	-	361.65 ± 130.64	-	158.74 ± 109.25	-
Adam	16.85 ± 10.32	-	720.15 ± 123.99	-	168.55 ± 108.67	-
Extrinsic parameters	q		q		q	
Batchsize	-0.99 ± 0.003		-1.13 ± 0.01		-1.35 ± 0.08	
No. of GPUs	-0.99 ± 0.004		-1.029 ± 0.001		-0.74 ± 0.001	
Constant term	C		C		C	
	3.703 ± 0.017		12.677 ± 0.038		1.930 ± 0.122	

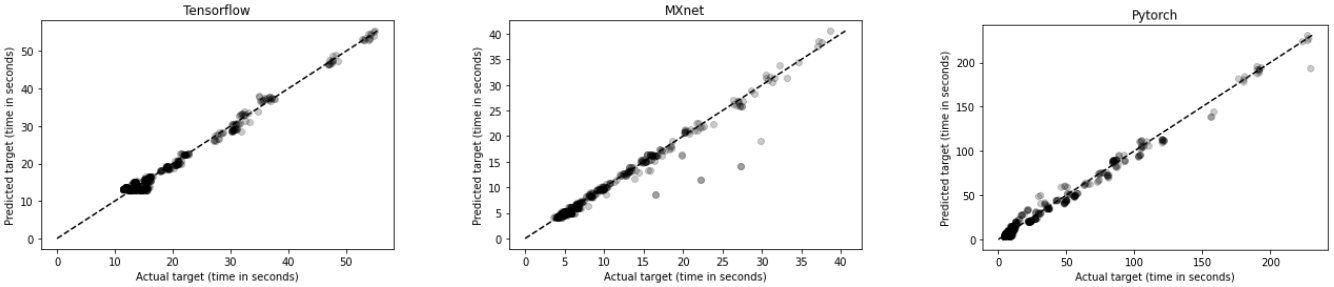
TABLE 4. Derived intrinsic and extrinsic parameters from the differential evolution-optimized performance models for the three deep learning frameworks using L2 regularisation. Parameters are given as the mean and standard deviation over ten fits. *a* and *p* represent coefficients and powers, respectively, of a term representing an intrinsic parameter, whereas *q* is power in a multiplicative term representing an extrinsic (scaling) parameter.

	Mxnet		Pytorch		TensorFlow	
Intrinsic parameters	a	p	a	p	a	p
Filter size	6.27 ± 0.59	0.36 ± 0.01	6.07 ± 1.59	0.89 ± 0.05	8.39 ± 0.37	0.77 ± 0.01
Kernel size	4.44 ± 0.65	0.50 ± 0.04	4.84 ± 1.90	2.02 ± 0.24	6.59 ± 0.29	2.04 ± 0.03
Pool size	4.69 ± 0.33	0.52 ± 0.03	3.23 ± 0.83	1.55 ± 0.45	6.70 ± 0.67	1.98 ± 0.05
Learning rate	3.62 ± 0.41	-0.04 ± 0.003	3.75 ± 1.70	-0.27 ± 0.02	4.40 ± 0.60	-0.22 ± 0.007
Stride	4.51 ± 0.40	-0.99 ± 0.11	2.92 ± 1.54	-0.83 ± 1.42	4.13 ± 0.59	2.46 ± 0.10
Dropout probability	4.20 ± 0.67	-0.35 ± 0.05	35.92 ± 1.15	-5.00 ± 0.00	4.46 ± 0.43	-1.94 ± 0.07
Same	2.66 ± 0.51	-	2.08 ± 0.84	-	1.90 ± 0.58	-
Valid	1.50 ± 0.43	-	-0.57 ± 1.56	-	0.49 ± 0.71	-
Sigmoid	2.18 ± 0.45	-	2.32 ± 1.15	-	1.41 ± 0.41	-
Relu	1.52 ± 0.33	-	3.21 ± 1.35	-	1.85 ± 0.64	-
Tanh	2.29 ± 0.39	-	2.93 ± 1.93	-	1.99 ± 0.71	-
MNIST	5.48 ± 0.52	-	3.37 ± 1.35	-	1.99 ± 0.72	-
Fashion-MNIST	7.73 ± 0.36	-	3.42 ± 1.56	-	2.28 ± 0.72	-
CIFAR-10	1.00 ± 0.02	-	1.89 ± 1.00	-	1.63 ± 0.69	-
SGD	2.31 ± 0.36	-	2.16 ± 1.00	-	1.73 ± 0.46	-
Adam	1.78 ± 0.41	-	3.42 ± 1.45	-	2.01 ± 0.85	-
Extrinsic parameters	q		q		q	
Batchsize	-0.87 ± 0.005		-1.00 ± 0.007		-1.19 ± 0.01	
No. of GPUs	-1.07 ± 0.007		-1.01 ± 0.004		-0.74 ± 0.005	
Constant term	C		C		C	
	3.45± 0.024		1.03 ± 0.07		12.62 ± 0.05	

coefficients, *Tanh* takes more training time than *Relu* and *Sigmoid* in MXnet and TensorFlow frameworks, while Relu takes maximum time with Pytorch. Also, in terms of dataset coefficients, the Fashion-MNIST dataset takes more training time than the MNIST and CIFAR-10 datasets in all three frameworks.

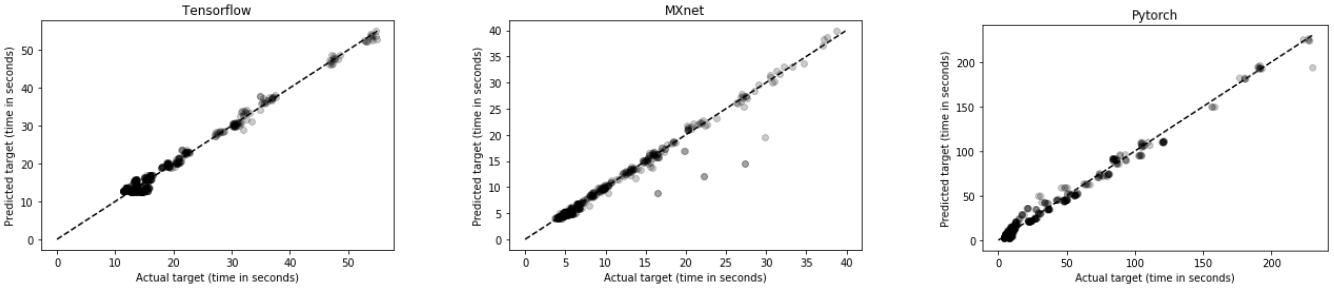
3) COMPARISON OF THE PROPOSED PERFORMANCE MODEL WITH BLACK BOX MODELS

We compared the proposed model with two standard black box models, i.e., random forest regressor and support vector regressor. Generally, the random forest has better prediction accuracy due to its ensemble learning shown



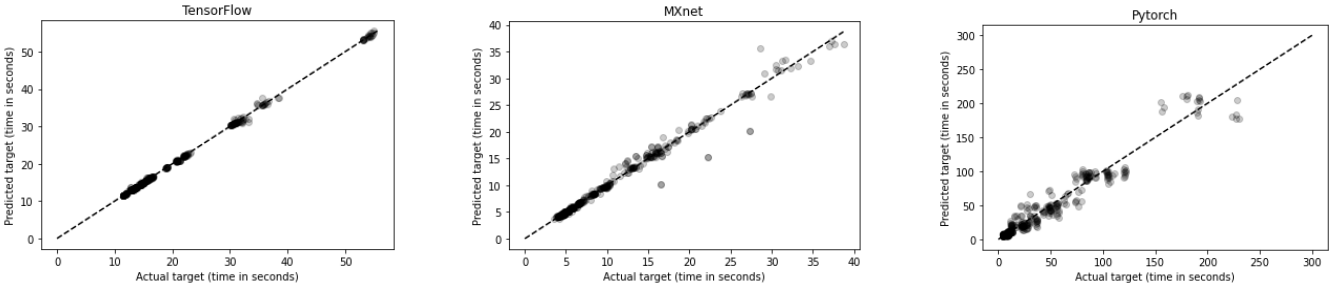
(a) TensorFlow (b) MXnet (c) Pytorch

FIGURE 3. The proposed performance model predicted and measured times in three deep learning frameworks using differential evolution algorithm.



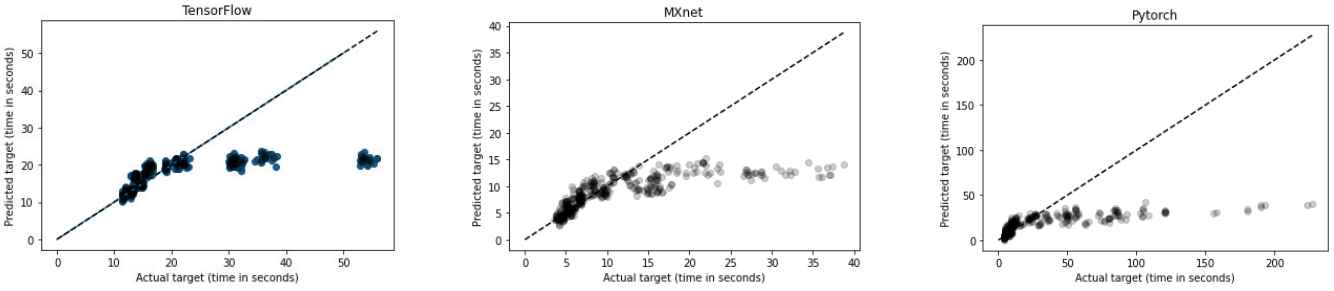
(a) TensorFlow (b) MXnet (c) Pytorch

FIGURE 4. The proposed performance model predicted and measured times in three deep learning frameworks using differential evolution algorithm using regularisation.



(a) TensorFlow (b) MXnet (c) Pytorch

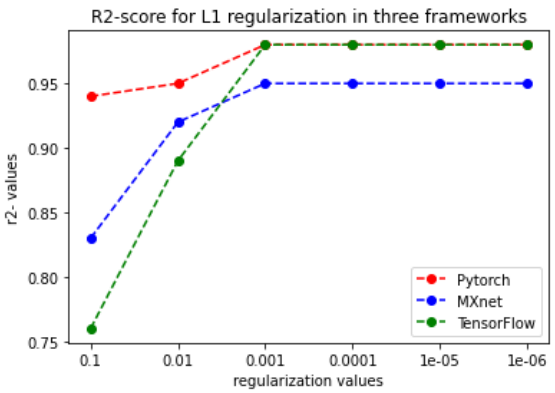
FIGURE 5. Random forest regressor predicted and measured times in three deep learning frameworks.



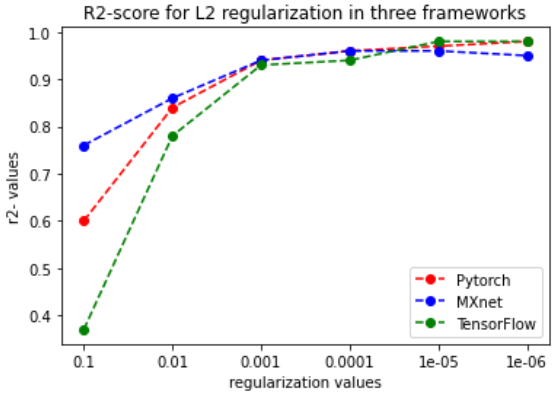
(a) TensorFlow (b) MXnet (c) Pytorch

FIGURE 6. Support vector regressor predicted and measured times in three deep learning frameworks.

in figure 5. The result shows a good linear fit compared to the differential evolution algorithm with and without regularisation. However, the drawback of the random forest regressor is that it cannot give any insights into its internal working mechanism. support vector regressor regression is a non-parametric technique because it depends on kernel

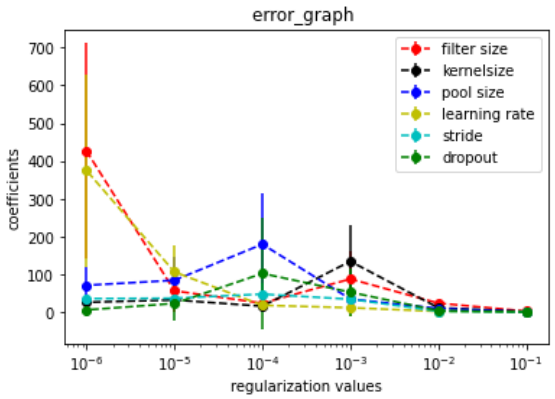


(a) R2 values with different regularisation values in three different frameworks using L1 regularisation

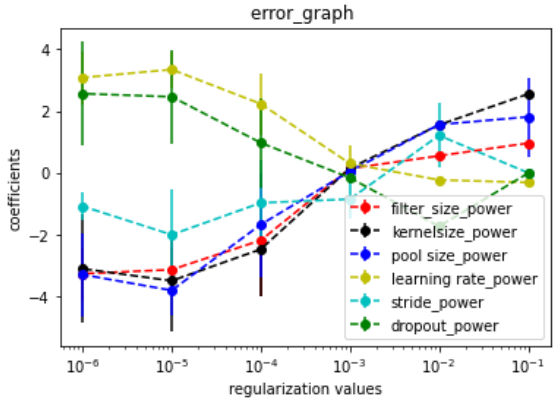


(b) R2 values with different regularisation values in three different frameworks using L2 regularisation.

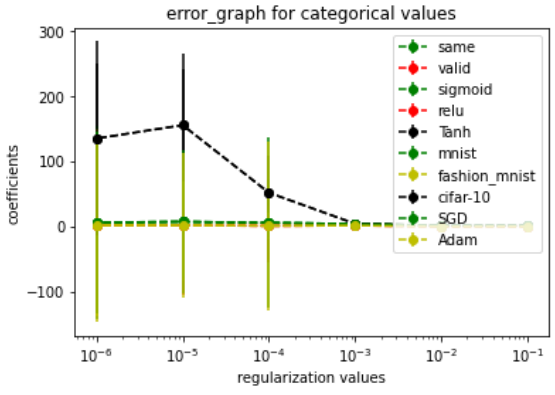
FIGURE 7. **Effect of regularisation.**



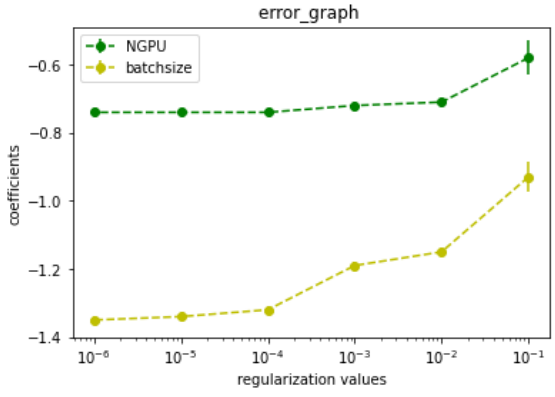
(a) TensorFlow



(b) MXnet



(c) Pytorch



(d) Pytorch

FIGURE 8. **Effect of regularisation, with model coefficients plotted against regularisation parameter. Constant coefficients of intrinsic parameters are plotted in (a), the power coefficients of intrinsic parameters are shown in (b), coefficients of categorical intrinsic parameters in (c), with powers of extrinsic parameters in (d).**

functionality. It is more productive in high-dimensional spaces. Figure 6 shows the predicted and measured times of the support vector regressor. The result shows a poor fit for all the deep learning frameworks compared with the random forest regressor and differential evolution algorithm with and without regularisation. We evaluate the fits using the mean absolute percentage error between predicted execution time and actual times, as shown in Table 6. Note that the performance of our proposed model is slightly inferior to the random forest. However, the proposed model can provide

TABLE 5. L1 and L2 regularisation results in terms of Mean Absolute Error (MAE), Mean Squared Error (MSE), Root Mean Squared Error (RMSE).

	L1 Regularisation			L2 Regularisation		
	MXnet	PYtorch	TF	MXnet	PYtorch	TF
MAPE	8%	29%	13%	7%	27%	10%
MSE	105.74	450.93	220.16	103.37	443.35	201.81
RMSE	10.28	17.52	14.83	10.16	17.02	14.20

TABLE 6. Mean Absolute Percentage Error on predictions of the performance models on the 300 instances in the evaluation dataset in seconds.

	TensorFlow	MXnet	Pytorch
Differential evolution	5%	5%	12%
Differential evolution with regularisation	10%	7%	14%
Random forest	0.7%	3%	23%
Support vector machine	16%	21%	54%

TABLE 7. nGPUs scaling power in various frameworks, nGPUs represent number of GPUs.

Frameworks	nGPUs scaling power
TensorFlow	-0.74
MXnet	-0.99
Pytorch	-1.02

insights into the internal behaviour and scalability, which are impossible with a black box model such as a random forest.

F. SCALABILITY

Observing the coefficients q from table 3 and table 4, where q is power in a multiplicative term representing an extrinsic parameter. The extrinsic parameter coefficients are consistent in the proposed performance model with and without regularisation. As shown in Table 7, -1 indicates ideal scaling, in which case the time is inversely proportional to the number of GPUs. The coefficients in Pytorch and MXnet frameworks show better scaling performance than TensorFlow. In TensorFlow, the value -0.73 is less than -1, indicating sub-optimal scaling.

The accuracy of the performance model in predicting training times in different deep learning frameworks over CIFAR-10 is shown in Table 8. The deep learning models' accuracy in the classification task used in this study varies significantly between the different deep learning frameworks. These accuracy values indicate how well the performance model can predict training times in each deep learning framework. PyTorch demonstrates the highest accuracy in predictions, while TensorFlow has a lower accuracy than MXNet and PyTorch.

V. CONCLUSION AND FUTURE WORKS

In this work, we have developed a generic performance model for deep learning applications in a distributed environment

TABLE 8. Accuracy of the performance model in different deep learning frameworks over CIFAR-10.

	MXNet	PyTorch	TensorFlow
Accuracy	78.60	91.66	71.05

with a generic expression of the application execution time that considers the influence of both intrinsic and extrinsic factors. We also formulated the proposed model as a global optimisation problem and solved it using regularisation on a cost function and differential evolution algorithm to find the best-fit values of the constants in the generic expression. Our proposed model has been evaluated on three widely used deep learning frameworks: TensorFlow, MXnet, and Pytorch. The results have shown that our model can provide accurate performance predictions and interpretability. Moreover, the experimental results reveal that MXnet and Pytorch demonstrate superior scalability performance compared to TensorFlow. Furthermore, our proposed method with regularisation has proven to optimise the vector constants and reduce high variance in intrinsic parameters. This model can be implemented in any distributed deep learning framework without necessitating any code modifications, providing insights into the factors influencing deep learning application performance and scalability. In future, we plan to evaluate the model's performance on various deep learning frameworks to assess its generalisation capability.

REFERENCES

[1] P. Ballester and R. M. Araujo, "On the performance of GoogLeNet and AlexNet applied to sketches," in *Proc. 30th AAAI Conf. Artif. Intell.*, 2016, pp. 1–5.

[2] S. Targ, D. Almeida, and K. Lyman, "ResNet in ResNet: Generalizing residual architectures," 2016, *arXiv:1603.08029*.

[3] L. Wang, S. Guo, W. Huang, and Y. Qiao, "Places205-VGGNet models for scene recognition," 2015, *arXiv:1508.01667*.

[4] N. Aloysius and M. Geetha, "A review on deep convolutional neural networks," in *Proc. Int. Conf. Commun. Signal Process. (ICCSP)*, Apr. 2017, pp. 588–592.

[5] F. Yan, O. Ruwase, Y. He, and T. Chilimbi, "Performance modeling and scalability optimization of distributed deep learning systems," in *Proc. 21st ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, Aug. 2015, pp. 1355–1364.

[6] Y. Oyama, A. Nomura, I. Sato, H. Nishimura, Y. Tamatsu, and S. Matsuoka, "Predicting statistics of asynchronous SGD parameters for a large-scale distributed deep learning system on GPU supercomputers," in *Proc. IEEE Int. Conf. Big Data (Big Data)*, Dec. 2016, pp. 66–75.

[7] M. Song, Y. Hu, H. Chen, and T. Li, "Towards pervasive and user satisfactory CNN across GPU microarchitectures," in *Proc. IEEE Int. Symp. High Perform. Comput. Archit. (HPCA)*, Feb. 2017, pp. 1–12.

[8] S. Shi, Q. Wang, P. Xu, and X. Chu, "Benchmarking state-of-the-art deep learning software tools," in *Proc. 7th Int. Conf. Cloud Comput. Big Data (CCBD)*, Nov. 2016, pp. 99–104.

[9] S. Shi, Q. Wang, and X. Chu, "Performance modeling and evaluation of distributed deep learning frameworks on GPUs," in *Proc. IEEE 16th Int. Conf. Dependable, Autonomic Secure Comput., 16th Int. Conf. Pervasive Intell. Comput., 4th Int. Conf. Big Data Intell. Comput. Cyber Sci. Technol. Congr.*, Aug. 2018, pp. 949–957.

[10] R. M. Rakshith, V. Lokur, P. Hongal, V. Janamatti, and S. Chickerur, "Performance analysis of distributed deep learning using Horovod for image classification," in *Proc. 6th Int. Conf. Intell. Comput. Control Syst. (ICICCS)*, May 2022, pp. 1393–1398.

[11] Z. Lin, X. Chen, H. Zhao, Y. Luan, Z. Yang, and Y. Dai, "A topology-aware performance prediction model for distributed deep learning on GPU clusters," in *Proc. IEEE Int. Conf. Big Data (Big Data)*, Dec. 2020, pp. 2795–2801.
[12] T. Kavarakuntla, "Performance modelling for scalable deep learning," Ph.D. thesis, Dept. Comput. Math., Manchester Metropolitan Univ., Manchester, U.K., 2023.
[13] S. Pllana, S. Benkner, F. Xhafa, and L. Barolli, "Hybrid performance modeling and prediction of large-scale computing systems," in *Proc. Int. Conf. Complex, Intell. Softw. Intensive Syst.*, 2008, pp. 132–138.
[14] T. Fahringer, S. Pllana, and J. Testori, "Teuta: Tool support for performance modeling of distributed and parallel applications," in *Proc. Int. Conf. Comput. Sci.* Cham, Switzerland: Springer, 2004, pp. 456–463.
[15] T. Chilimbi, Y. Suzue, J. Apacible, and K. Kalyanaraman, "Project Adam: Building an efficient and scalable deep learning training system," in *Proc. 11th USENIX Symp. Operating Syst. Design Implement.*, 2014, pp. 571–582.
[16] H. Qi, E. R. Sparks, and A. Talwalkar, "Paleo: A performance model for deep neural networks," in *Proc. Int. Conf. Learn. Represent.*, 2016.
[17] H. Kim, H. Nam, W. Jung, and J. Lee, "Performance analysis of CNN frameworks for GPUs," in *Proc. IEEE Int. Symp. Perform. Anal. Syst. Softw. (ISPASS)*, Apr. 2017, pp. 55–64.
[18] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, and M. Isard, "TensorFlow: A system for large-scale machine learning," in *Proc. 12th USENIX Symp. Operating Syst. Design Implement.*, 2016, pp. 265–283.
[19] F. Seide and A. Agarwal, "CNTK: Microsoft's open-source deep-learning toolkit," in *Proc. 22nd ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, 2016, p. 2135.
[20] R. Al-Rfou, G. Alain, A. Almahairi, C. Angermueller, D. Bahdanau, N. Ballas, F. Bastien, J. Bayer, A. Belikov, and A. Belopolsky, "Theano: A Python framework for fast computation of mathematical expressions," 2016, *arXiv:1605.02688*.
[21] A. A. Awan, K. Hamidouche, J. M. Hashmi, and D. K. Panda, "S-Caffe: Co-designing MPI runtimes and Caffe for scalable deep learning on modern GPU clusters," in *Proc. 22nd ACM SIGPLAN Symp. Princ. Pract. Parallel Program.*, 2017, pp. 193–205.
[22] R. Collobert, S. Bengio, and J. Mariéthoz, "Torch: A modular machine learning software library," Idiap, Martigny, Switzerland, Tech. Rep., 2002.
[23] T. Kavarakuntla, L. Han, H. Lloyd, A. Latham, and S. B. Akintoye, "Performance analysis of distributed deep learning frameworks in a multi-GPU environment," in *Proc. 20th Int. Conf. Ubiquitous Comput. Commun.*, Dec. 2021, pp. 406–413.
[24] D. Didona, F. Quaglia, P. Romano, and E. Torre, "Enhancing performance prediction robustness by combining analytical modeling and machine learning," in *Proc. 6th ACM/SPEC Int. Conf. Perform. Eng.*, Jan. 2015, pp. 145–156.
[25] R. Storn and K. Price, "Differential evolution—A simple and efficient heuristic for global optimization over continuous spaces," *J. Global Optim.*, vol. 11, no. 4, pp. 341–359, Dec. 1997, doi: 10.1023/A:1008202821328.
[26] C.-Y. Lee and C.-H. Hung, "Feature ranking and differential evolution for feature selection in brushless DC motor fault diagnosis," *Symmetry*, vol. 13, no. 7, p. 1291, Jul. 2021. [Online]. Available: https://www.mdpi.com/2073-8994/13/7/1291
[27] W. Yang, E. M. D. Siriwardane, R. Dong, Y. Li, and J. Hu, "Crystal structure prediction of materials with high symmetry using differential evolution," *J. Phys., Condens. Matter*, vol. 33, no. 45, Nov. 2021, Art. no. 455902.
[28] S. Saha and R. Das, "Exploring differential evolution and particle swarm optimization to develop some symmetry-based automatic clustering techniques: Application to gene clustering," *Neural Comput. Appl.*, vol. 30, no. 3, pp. 735–757, Aug. 2018, doi: 10.1007/s00521-016-2710-0.
[29] Y.-H. Li, J.-Q. Wang, X.-J. Wang, Y.-L. Zhao, X.-H. Lu, and D.-L. Liu, "Community detection based on differential evolution using social spider optimization," *Symmetry*, vol. 9, no. 9, p. 183, Sep. 2017. [Online]. Available: https://www.mdpi.com/2073-8994/9/9/183
[30] M. Baioletti, A. Milani, and V. Santucci, "Learning Bayesian networks with algebraic differential evolution," in *Parallel Problem Solving From Nature—PPSN XV*. Cham, Switzerland: Springer, 2018, pp. 436–448.
[31] M. F. Ahmad, N. A. M. Isa, W. H. Lim, and K. M. Ang, "Differential evolution: A recent review based on state-of-the-art works," *Alexandria Eng. J.*, vol. 61, no. 5, pp. 3831–3872, May 2022. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S111001682100613X
[32] D. Liu, D. Hong, S. Wang, and Y. Chen, "Genetic algorithm-based optimization for color point cloud registration," *Frontiers Bioeng. Biotechnol.*, vol. 10, p. 923736, 2022. [Online]. Available: https://www.frontiersin.org/articles/10.3389/fbioe.2022.923736
[33] M. Baioletti, G. Di Bari, A. Milani, and V. Poggioni, "Differential evolution for neural networks optimization," *Mathematics*, vol. 8, no. 1, p. 69, Jan. 2020. [Online]. Available: https://www.mdpi.com/2227-7390/8/1/69
[34] N. Ikushima, K. Ono, Y. Maeda, E. Makihara, and Y. Hanada, "Differential evolution neural network optimization with individual dependent mechanism," in *Proc. IEEE Congr. Evol. Comput. (CEC)*, Jun. 2021, pp. 2523–2530.
[35] R. A. Venkat, Z. Oussalem, and A. K. Bhattacharya, "Training convolutional neural networks with differential evolution using concurrent task apportioning on hybrid CPU-GPU architectures," in *Proc. IEEE Congr. Evol. Comput. (CEC)*, Jun. 2021, pp. 2567–2576.
[36] K. Fleetwood, "An introduction to differential evolution," in *Proc. MASCOS*, 2004, pp. 785–791.
[37] S. Park, J. Lee, and H. Kim, "Hardware resource analysis in distributed training with edge devices," *Electronics*, vol. 9, no. 1, p. 28, Dec. 2019.
[38] A. Khan, A. Sohail, U. Zahoora, and A. S. Qureshi, "A survey of the recent architectures of deep convolutional neural networks," 2019, *arXiv:1901.06032*.
[39] Y. LeCun and C. Cortes. (2020). *The MNIST Database of Handwritten Digits*. [Online]. Available: http://yann.lecun.com/exdb/mnist/
[40] L. Deng, "The MNIST database of handwritten digit images for machine learning research [best of the web]," *IEEE Signal Process. Mag.*, vol. 29, no. 6, pp. 141–142, Nov. 2012.
[41] H. Xiao, K. Rasul, and R. Vollgraf, "Fashion-MNIST: A novel image dataset for benchmarking machine learning algorithms," 2017, *arXiv:1708.07747*.
[42] F. O. Giuste and J. C. Vizcarra, "CIFAR-10 image classification using feature ensembles," 2020, *arXiv:2002.03846*.

TULASI KAVARAKUNTLA received the master's degree in computer science from JNTUH, India, in 2007. She is currently pursuing the Ph.D. degree with the Computing Department, Manchester Metropolitan University, with a focus on scalable deep learning. Her research interests include machine learning and developing performance models for scalable deep learning in a distributed environment.

LIANGXIU HAN received the Ph.D. degree in computer science from Fudan University, Shanghai, China, in 2002. She is currently a Professor in computer science with the Department of Computing and Mathematics, Manchester Metropolitan University. She is also a Principal Investigator or a Co-PI on a number of research projects in the research areas mentioned above. Her research interests include the development of novel big data analytics and the development of novel intelligent architectures that facilitates big data analytics (e.g., parallel and distributed computing and cloud/service-oriented computing/data intensive computing) and applications in different domains using various large datasets (biomedical images, environmental sensor, network traffic data, and web documents).

HUW LLOYD (Member, IEEE) received the B.Sc. degree in physics from Imperial College, London, U.K., and the Ph.D. degree in astrophysics from The University of Manchester, U.K. He is currently a Senior Lecturer with the Department of Computing and Mathematics, Manchester Metropolitan University. His research interests include theoretical and applied topics in machine learning, evolutionary computation, combinatorial, and continuous optimization.

ANNABEL LATHAM (Senior Member, IEEE) is currently a Senior Lecturer with the Department of Computing and Mathematics, Manchester Metropolitan University. Her research interests under the Intelligent Systems Laboratory include conversational agents, intelligent tutoring systems, affective computing, the ethics of AI in education, user profiling, and computational intelligence. She was also the Past Chair of IEEE U.K. and Ireland Women in Engineering (2019–2022), a Winner of 2019 Region 8 WIE Group of the Year, and the Chair of CIS Education Strategic Planning Subcommittee. She is a U.K. STEM Ambassador.

ANTHONY KLEEREKOPER received the M.Eng. degree in information systems engineering from Imperial College London, in 2009, and the Ph.D. degree from The University of Manchester, in 2013, for work on distributed construction of load balanced routing trees in many to one wireless sensor networks. After two years as a Research Associate with The University of Manchester, he was appointed as a Lecturer in computer science with Manchester Metropolitan University, where he has been a Senior Lecturer, since 2018. He is a fellow of the Higher Education Academy. His research interests include varied and include data visualization, algorithms for processing data streams, agent-based modeling, and opinion dynamics.

SAMSON B. AKINTOYE received the Ph.D. degree in computer science from the University of the Western Cape, South Africa, in 2019. He is currently a Research Associate with the Department of Computing and Mathematics, Manchester Metropolitan University, U.K. His current research interests include parallel and distributed computing, deep learning, and cloud computing.