Evaluating Machine Learning Models for Disparate Computer Systems Performance Prediction

Amit Mankodi*, Amit Bhatt†, Bhaskar Chaudhury†, Rajat Kumar* and Aditya Amrutiya*
*Dirubhai Ambani Institute of Information and Communication Technology, Gandhinagar, India
Email: mankodiamit@gmail.com, rajat.tech.002@gmail.com, amrutiyaaditya@gmail.com
†Dirubhai Ambani Institute of Information and Communication Technology, Gandhinagar, India

***Abstract*—Performance prediction is an active area of research due to its applications in the advancements of hardware-software co-development. Several empirical machine-learning models such as linear models, tree-based models, neural network etc are used for performance prediction each having different prediction accuracy. Furthermore, the prediction model's accuracy may differ depending on performance data collected for different software types (compute-bound, memory-bound) and different hardware (simulation-based or physical systems). We have studied fourteen machine-learning models on simulation-based hardware and physical systems by executing several benchmark programs with different computation and data access patterns. Our results show that the tree-based machine-learning models outperform all other models with median absolute percentage error (MedAPE) of less than 5% followed by bagging and boosting models that help to improve weak learners. We have also observed that prediction accuracy is higher on simulation-based hardware due to its deterministic nature as compared to physical systems. Moreover, in physical systems, prediction accuracy of memory-bound algorithms is higher as compared to compute-bound algorithms due to manufacturer variability in processors.**

. ***Index Terms*—machine learning models; computer architecture; performance prediction; metrics;**

I. Introduction

Performance modeling is an active area for research. It aids us with a tool for estimation of performance (runtime) of a given application on an unknown hardware system or estimation of new algorithm on a given hardware system [1]. It is useful for software developers and system architects for choosing the most optimal strategy of application and configuration of the underlying hardware systems. The estimation of runtime or performance of an application is determined by feature of underlying hardware on which it is executed such as processor and memory features. The performance of applications requiring more computation will depend more on processor clock speed, number of cores and cache, etc. On the other hand, applications requiring higher data access will depend more on memory clock, type, etc.

Several research works have focused on performance prediction. The study in [2] evaluates machine learning models for modeling the performance of scientific applications with skewed and irregular domain configurations on four leadership class HPC platforms. Work in [3] deals with predicting optimal cloud configurations for HPC before deployment. The authors have predicted both provider side cloud performance and user side application profile with prediction errors below 12% for cloud and 30% for user side. [4] also uses performance modeling to define cloud-based auto-scaling policies to minimize the QoS violations. The work in [5] predicts application performance by using cross-core performance interference on multi-core processors. The prediction of power and performance on heterogeneous systems using neural networks is studied in [6]. The survey paper [7] also compares tools for performance prediction for a variety of different architectures and applications. The prediction accuracy rate of 97.5% was achieved in [8] using multiple neural networks and PCA on SPEC CPU2006 and SPEC CPU2017 benchmarks.

We have identified questions from related work, which are: (i) Reference work uses several machine-learning algorithms, which ones have better prediction accuracy and why? (ii) What inferences can be drawn from the performance modeling of memory-bound and compute-bound applications? (iii) Does the machine learning model behave similarly for simulation-based hardware as compared to physical hardware systems?. In this paper, we address these questions for a better understanding of the performance model using different machine learning models.

The remainder of the paper is organized as follows: Section II describes the computer hardware selection and dataset. Section III describes machine learning models that we have used for the study. Section IV provides three scenarios: (a) Overall performance of models on each dataset and discussion on the inferences that can be drawn. (b) Evaluation of results concerning compute and memory-bound applications. (c) Analysis of performance with respect to physical and simulated systems for each model. Finally, section V provides concluding remarks and work that we plan to continue.

II. DataSet Preparation

A. Algorithms Selection for Workload

For testing the applicability and accuracy of the performance prediction models, we had used two categories of applications. Compute bound applications consist of the Montecarlo to calculate the value of PI, MSER algorithm from SD-VBS [9], miniFE [10] from the Mantevo benchmark suite, and EP algorithm from the NAS parallel benchmark [11]. The memory-bound application consists of Matrix Multiplication, Tracking algorithm from SD-VBS [9], and MG algorithm from the NAS benchmark [11]. These applications were executed on

TABLE I
COMPUTER HARDWARE MODELS BUILT IN GEM5 SIMULATOR

H/W Class	ISA	CPU Speed GHz	Cores	Mem Type	Mem Access MHz*	L1,L3 Cache Size**	Cnt
1	X86	2-3.5	2-18	DDR4	2400-2666	32,64	50
2	X86	2.8-4.7	1-8	DDR3	1600-1866	16,8	60
3	ARM	1.7-2	4,8	DDR4	1866	32,8	15
4	ARM	1-2.7	2-8	LPDDR2	400-1866	4,3	70
5	ARM	1.1-2.34	1-4	LPDDR3	1600-1866	32,4	35
6	X86	1.3-3.5	2,4	DDR3	1600	32,8	60
7	X86	1.7-3.5	2-18	DDR4	1866-266	32,16	95
8	X86	1.3-3.5	2,4	LPDDR3	1600-2133	32,8	90

*Memory Size range 1GB to 8GB
**L1 Cache Size is in kB and L3 Cache Size is in MB

TABLE II
PHYSICAL COMPUTER HARDWARE MODELS

Sr	ISA	CPU Speed GHz	Cores	Mem Type	Mem Access MHz	Mem Size GB	L1-L3 Cache Size*
1	X86	3.2	4	DDR3	1600	4	32,6
2	X86	3	2	DDR3	1600	4	32,4
3	X86	2.6	2	DDR3	1066	4	32,4
4	X86	3.2	4	DDR3	1600	4	32,6
5	X86	3.2	4	DDR4	2400	4	32,6
6	X86	3.2	4	DDR4	2666	16	32,12
7	X86	2.4	12	DDR4	2133	64	32,15
8	X86	2	16	DDR3	1600	32	32,20

*L1 Cache Size is in kB and L3 Cache Size is in MB
Configuration taken from following models
1. Intel Core i56500. 2. Intel Core i76500U 3. Intel Core i7620M 4. Intel Core i53470 5. Intel Core i56500 6. Intel Core i78700 7. Intel Xeon E52620v3 8. Intel Xeon E52640v2

the simulated environment and physical systems from which the runtime was collected.

B. Computer Hardware Selection

To execute the selected applications, we have selected computer systems that describe the class of computers commonly used today. We have chosen two server systems with Intel Xeon processors with many cores and large memory, three Intel Core i7 systems, and three Intel Core i5 systems with the configurations listed in table II. For the simulation dataset, we have used the Gem5 simulator, a widely accepted simulator for architectural research. We have built 120 hardware models based on the ARM instruction set (ISA) and 355 hardware models based on the X86 ISA. We have used nine hardware model features for configuration of Gem5 models as shown in table I with values from the real memory and processor models commonly used in today's computers.

We have executed applications listed in "Scientific Applications" column on each hardware type listed in "System Type" column in table III. For each algorithm, we have executed the number of processes from one to two times the number of cores in the system; that is, we have executed 24 processes on a system with 12 cores. This is to take advantage of systems with hyper-threading. We have extracted runtimes for each execution from the benchmark logs.

III. MACHINE LEARNING MODELS

Performance modeling using empirical evaluation can be done using supervised machine learning algorithms. A labelled dataset is used to make algorithm learn the function $f_{(x)}$ which maps inputs X_i to an output Y_i. Here X_i may be a multidimensional vector or tuple while the output Y_i is a single numerical value in our case. So, the different ML models are used to derive that function $f_{(x)}$ which gives the most accurate predicted Y value. In our case, each sample X_i is a multidimensional tuple corresponding to processor and memory characteristics as shown in table II for physical systems and table I for simulated systems. Y_i is runtime or performance of algorithm. The overview of the models that we use in our paper is as follows.

TABLE III
DATASET INFORMATION AND TYPE

Scientific Applications	System Type	Memory/Compute Bound (MB/CB)	Data Points
Matrix Mult.	Physical	MB	280
Matrix Mult.	Simulated	MB	1900
Montecarlocalcpi	Simulated	CB	1425
Montecarlocalcpi	Physical	CB	224
MSER	Simulated	CB	475
Mantevo miniFE	Physical	CB	124
NPB EP	Physical	CB	108
NPB MG	Physical	MB	108
Tracking	Simulated	MB	475

A. Support Vector Regressor (svr)

Support Vector Regression [12] performs function approximation (linear or non-linear) by formulating a constrained optimization problem. For non-linear functions, the kernel trick is used to map X_i into a higher dimensional space called as kernel space to get better predictions. Svr uses $\epsilon - insensitive$ loss function and penalizes predictions that are far away from the actual output.

B. Multiple Linear Regression (lr)

Simple linear regression has one dimensional input whereas a multiple linear regression has multidimensional input X_i and both having one dimensional output Y_i. A simple linear regression model would have the form: $y = \alpha + x\beta + \epsilon$ and a multi-variable or multiple linear regression model takes the form: $y = \alpha + x_1\beta_1 + x_2\beta_2 + ... + x_k\beta_k + \epsilon$,where y is a dependent variable, x is a single value in the simple regression model, and $x_1, x_2, \ldots, x_k$ are the multiple values in the multiple regression model.

C. Ridge Regression (rr)

Ridge regression [13] is an extension for linear regression when a penalty term is added to the loss function to avoid overfitting and when there is multi-collinearity among the features. Loss function $L = \Sigma(\hat{Y}_i - Y_i)^2 + \lambda(\Sigma\beta^2)$ is used, where λ is a penalty term for regularization and β are the coefficients to be estimated.

D. K Nearest Neighbors (knn)

K Nearest Neighbors Regression [14] belongs to the class of instance-based methods, where the model is not built explicitly during the training phase but determined only during

the prediction of value. KNN uses the concept of neighbor instances based on the distance of predicting X and predefined X. It selects the user-defined number of neighbors and average their actual Y values to give us the predicted Y value.

E. Gaussian Process Regressor (gpr)

Gaussian process regression [15] model implements Gaussian processes for regression purposes. The algorithm finds the normal distribution which maximizes the log marginal likelihood using an optimizer parameter. The Gaussian Process gives the predictive variance estimate around the prediction as well as having a clear probabilistic interpretation.

F. Decision Trees Regressor (dt)

Decision Tree regression [16] belongs to the class of recursive partitioning methods. The decision tree is used to build regression models that are similar to a tree data-structure. The data is split on the attribute or feature which provides the highest information gain. The depth of the final tree can be limited by an argument called max_depth. For continuous variables, the impurity measure is defined by the weighted Mean Squared Error of the children nodes.

G. Random Forest Regressor (rf)

Random Forest [17] is a bagging approach that builds a large number of decision trees for the same dataset. It reduces the high variance part of decision trees for better predictions. Each tree has a different associated rule of dividing the dataset into smaller subsets. It outputs the mean prediction of the individual trees that it builts. This technique of using multiple models to obtain higher predictive performance is called model ensembling.

H. Extremely Randomized Trees (etr)

Extremely randomized trees [18] or Extra Trees is like a Random Forest. But, it does not bootstrap data items (meaning it samples without replacement), and nodes are split on random splits and not the best split. In Extra Trees, randomness does not come from bootstrapping of the data, but rather it comes from the random splits of all the observations. Also, It runs faster than random forest.

I. Gradient Boosting Regressor (gbr)

Gradient boosting regression [19] is a boosting technique where weak learners are combined to get a strong learner. Decision Trees are used as weak learners. Trees are formed in a greedy manner while choosing the best split using information gain. Trees follow an additive approach one at a time. Gradient Boosting uses gradients of the error function to add a decision tree. This is done by modifying the parameters of the tree and moving in the direction to minimize the error.

J. XGBoost (xgb)

XGBoost [20] stands for eXtreme Gradient Boosting. It is an optimized variant of the gbr model practicing parallel programming, pruning of trees, and regularization to avoid model overfitting. This algorithm has built-in cross-validation for each epoch. Furthermore, the model handles missing data values automatically which makes this it a sparse aware model.

K. Deep Neural Networks

DNN is an artificial neural network with multiple hidden layers. This model estimates the relationship (whether linear or non-linear) between input and output. We used four DNN variants with a summary, as shown in table IV. We have used mean absolute error ($MAE(y, \hat{y}) = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y_i}|$) loss function for our DNN models. Rectified Linear Unit (ReLU) is used as an activation function in hidden layers for non-linearity, whereas the linear activation function is used for summing up the predicted performance value from the last hidden layer to the output layer. The number of epochs used is 100 for each variant.

TABLE IV
MODEL SUMMARY OF DNN VARIANTS

	dnn_1		**dnn_2**		**dnn_3**		**dnn_4**	
LayerType	Dense		Dense		Dense		Dense	
	Neurons	Params	Neurons	Params	Neurons	Params	Neurons	Params
Layer1	512	11776	512	11776	256	5888	512	11776
Layer2	1	513	512	262656	64	16488	128	65664
Layer3	N/A	N/A	512	262656	16	1040	32	4128
Layer4	N/A	N/A	1	513	4	68	8	264
Layer5	N/A	N/A	N/A	N/A	1	5	2	18
Layer6	N/A	N/A	N/A	N/A	N/A	N/A	1	3

IV. EXPERIMENTAL EVALUATION/RESULTS

In this section, we have performed different experiments to evaluate machine learning models described in section III using dataset from section II. We have used categorical one-hot encoding to convert categorical data from 'isa' and 'mem-type' attributes to real values. For data normalization, we have used $StandardScaler()$ function from sklearn library [21]. Furthermore, we have used ten-fold cross-validation using $ShuffleSplit()$ function [21] to assess the robust nature of the models.

We have trained each machine-learning model with 80% of the dataset, and prediction accuracy is measured with the remaining 20%. We have used R2 score, Median Absolute Percentage Error(MedAPE), and Mean Squared Error(MSE) metric to measure the accuracy of models. Low prediction MedAPE and MSE represent model has high prediction accuracy, whereas R2 score of 1 indicates model fits the data well. The negative R2 score indicates a worse model fit than a horizontal hyperplane.

We have evaluated machine learning models in three different scenarios. Firstly, the overall result for each model averaged over all nine datasets. Secondly, a comparison of model performance concerning the types of applications and

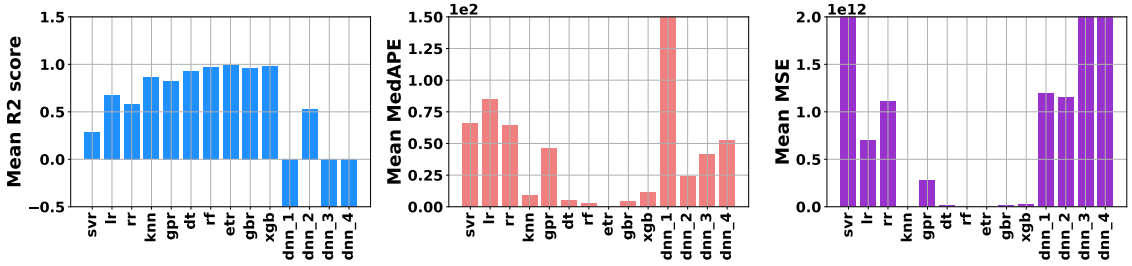


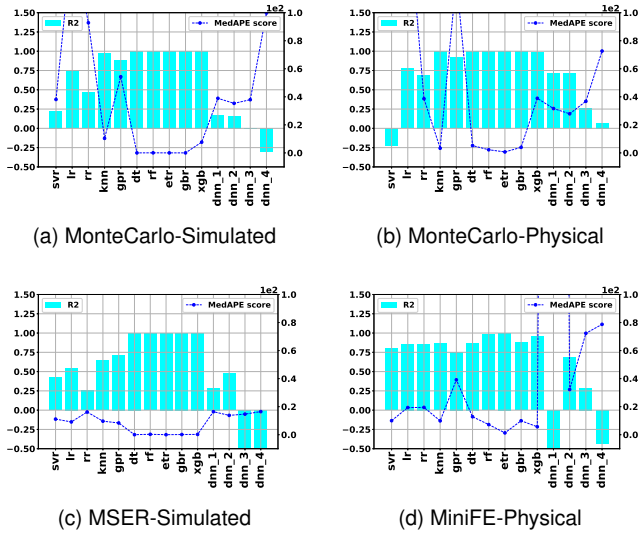
Fig. 1. Mean of R2, MedAPE and MSE values per model for all datsets

finally, a comparison of model performance between physical and simulated systems data is made.

1) Model comparison for performance prediction

Fig. 1 shows the mean value of all dataset R2 score and MedAPE score. We have observed that the order of model performances in decreasing order is: 'etr', 'rf', 'gbr', 'dt', 'xgb', 'knn', 'gpr', 'lr', 'rr', 'dnn_2', 'dnn_4', 'dnn_3', 'svr', 'dnn_1'. This is the relative order of performance that we get after comparing all 3 plots in Fig.1 and after taking a majority vote of all three metrics, namely R2, MedAPE, and MSE scores. The explanation for this behavior in performance concerning each model is explained below.

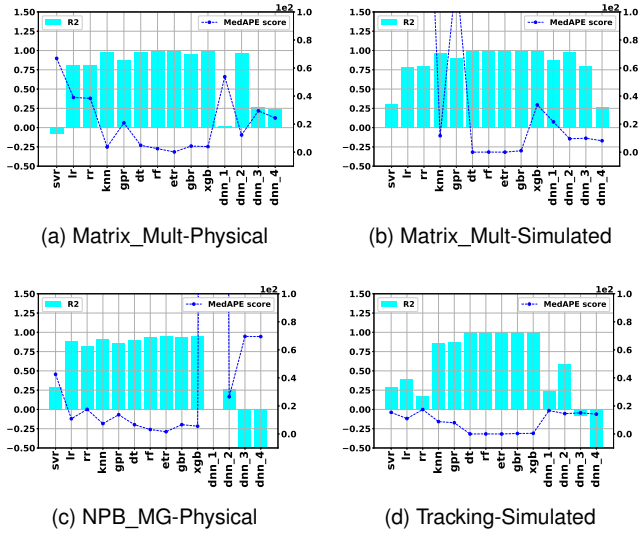
- ***svr***: Support Vector Regressor with mean R2 = 0.28 and mean MedAPE = 65.91% (Fig. 1), is not able to perform well because function to be approximated is non-linear and the distribution of each attribute is also skewed. We also observed that parameter C (regularization parameter) required value 1000 using grid search to provide better results. This means that a smaller margin hyperplane is chosen due to which the model is not able to perform well.
- ***lr and rr***: Linear Regressor with mean R2 = 0.68 (Fig.1), which have poor accuracy because the runtime is not linearly dependent on all the memory and processor features. The ridge regression regularises the overfitted linear model and can be observed in Fig. 1 where MedAPE for linear regression is 84.72% whereas for ridge regression is 64.02%.
- ***knn***: KNN Regressor with mean R2 = 0.86 and mean MedAPE = 9.44% (Fig.1) performs better than linear models and kernel svr. The parameters n_neighbors and metrics that play the important roles in performance are found using grid search. The best distance metric for our data is Minkowski, which is a generalization of both euclidean and manhattan distance. The good performance points to the data having feature similarities in the high dimensional vector space.
- ***gpr***: Gaussian Process Regressor with mean R2 = 0.83 and mean MedAPE = 46.29% (Fig.1) also performs better than linear models and kernel svr. But still, tree-based models are better. The reason for less performance can be explained by the fact that gpr assumes the Gaussian process prior, which is a multivariate gaussian distribution, and then learns the posterior distribution during training. This means that the distribution of our dataset is not following multivariate gaussian distribution and is not be tractable by the model.
- ***dt***: Decision Tree Regressor with mean R2 = 0.93 and mean MedAPE = 4.95% (Fig.1) performs better than all the models discussed above. The parameter max_depth of the decision tree is above 10 in most cases of grid search on each application data, which provides the model with a large number of decision rules to divide the data into smaller subsets. Due to the categorical characteristics of dataset features, the decision tree structure has given a good performance.
- ***rf and etr***: These two models are giving best performance prediction accuracy. Both algorithms are bagging approaches of the decision tree, so a large number of trees improve the accuracy of the model. Extra Trees Regressor with mean R2 = 0.99 and mean MedAPE = 0.69% (Fig.1) performs better than Random Forest Regressor with mean R2 = 0.97 and mean MedAPE = 2.58%. Extra Trees perform better than Random Forest in case the features are noisy. Since etr splits are random, more diversified trees are generated in this case, which strengthens the model.
- ***gbr and xgb***: Boosting algorithms are based on weak learners (high bias and low variance). Gradient Boosting Regressor with mean R2 = 0.96 and mean MedAPE = 4.02% (Fig.1) performs better than eXtreme Gradient Boosting with mean R2 = 0.97 and mean MedAPE = 11.57%. These boosting models reduce the bias to get good accuracy. But in our datasets, the base learner is outstanding and has a low bias in the case of the simple decision tree as well. Moreover, boosting tends to overfit the data. So the boosting algorithm works well but accuracy is lower than the bagging approaches.
- ***dnn***: Deep Neural Networks (with best model among the four variants (table IV) to be dnn_2) with mean R2 = 0.53 and mean MedAPE = 23.98% (Fig.1) are not able to perform well here. We believe that due to limited number of data points available (as can be seen in our case in table III), neural network models have lower accuracy than tree-based models.



(a) MonteCarlo-Simulated (b) MonteCarlo-Physical

(c) MSER-Simulated (d) MiniFE-Physical

Fig. 2. R2 score and MedAPE for Compute Bound Applications



(a) Matrix_Mult-Physical (b) Matrix_Mult-Simulated

(c) NPB_MG-Physical (d) Tracking-Simulated

Fig. 3. R2 score and MedAPE for Memory Bound Applications

2) Estimating the effect of performance with respect to Compute Bound and Memory Bound Applications

Figures 2 and 3 shows R2 and MedAPE scores on left and right y-axis with ranges $[-0.5, 1.5]$ and $[-10\%, 100\%]$ respectively for compute-bound and memory-bound algorithms from dataset (refer table III). The first observation from Fig. 2 and 3 is that five models which are 'dt', 'rf', 'etr', 'gbr' and, 'xgb' are giving above 0.75 mean R2 score and below 20% mean MedAPE score in nearly all scientific applications. These models are a good fit for datasets listed in table III. So for the next observations, we have considered only these five tree-based models for comparison of overall performance. The second observation that can be taken from Fig.4 is that, on an average taken over MedAPE values of the tree-based models, prediction accuracy in Memory-Bound applications

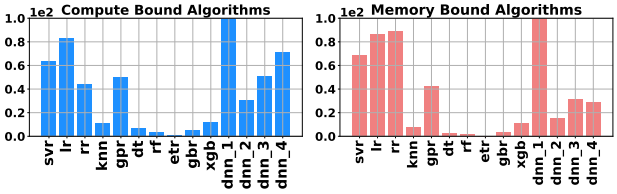
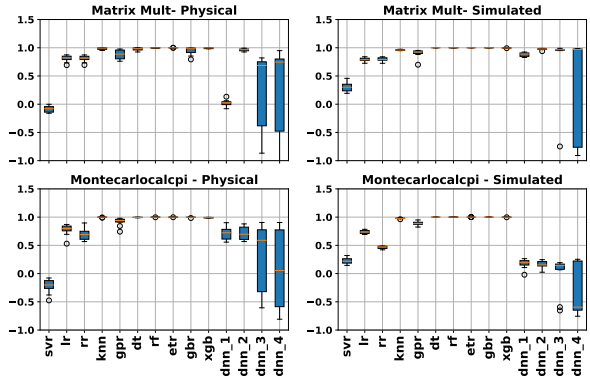
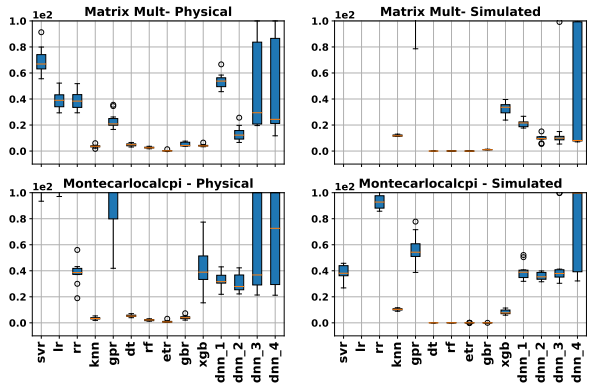


Fig. 4. Mean MedAPE for both CB and MB Applications



(a) R2 score



(b) Absolute Percentage Error

Fig. 5. Comparing model performance on Physical and Simulated Systems

(with 3.75% mean error) is slightly better than in Compute-Bound applications (with 5.58% mean error). This is because the processors have higher manufacturer variability due to complex architecture design than memory.

3) Comparing model performance on Physical and Simulated Systems

Fig. 5 shows R2 score and MedAPE gathered from 10-fold cross-validation for performance prediction on physical and simulated systems for Matrix Multiplication (memory-bound) and Motecarlocalcpi (compute-bound) algorithms. The models that do not show box-plot have much higher error falling outside the y-axis range. By comparing the performance of tree-based models, it is observed that the prediction accuracy on simulated systems (with Mean R2 = 0.999 and Mean MEdAPE = 4.20%) are slightly higher than physical systems (with Mean R2 = 0.989 and Mean MEdAPE = 7.16%). This is due to the larger variance in runtime for non-deterministic

physical systems than for deterministic simulated systems. Furthermore, we also, find that among the two applications, Matrix Multiplication (memory-bound) with Mean R2 = 0.98 and Mean MedAPE = 3.46% and Montecarlocalcpi (compute-bound) with Mean R2 = 0.99 and Mean MedAPE = 10.85%, when executed on physical systems, memory-bound applications perform better. This is because the higher variance in runtime contributed by manufacturer variability from the complex architecture of processor in the physical system results in higher error in compute-bound applications. On the other hand, among the same two applications- Matrix Multiplication with Mean R2 = 0.990 and Mean MedAPE = 6.71% and Montecarlocalcpi with Mean R2 = 0.9997 and Mean MedAPE = 1.69%, when executed on simulated system, compute-bound applications perform better. In the case of simulated systems, the Gem5 simulator, having a single out-of-order five stage pipeline CPU, causes lower variance in the runtime of compute-bound applications, resulting in higher prediction accuracy.

V. Conclusion And Future Work

The overall datasets of the experiment consist of nine different databases collected by executing tests of four scientific applications on 475 simulated models made in Gem5 simulator, while five scientific applications were executed on eight different physical computer systems. The conclusions that can be made from the experiments performed are as follows:

- The tree-based models including tree-based bagging models dt, etr and rf outperform all other machine learning models including the deep learning variants.
- The performance prediction on memory-bound applications have slightly better accuracy than compute-bound applications due to manufacturer variability.
- After exhaustive comparison of physical and simulated system applications, it is concluded that models have higher accuracy on simulated systems.
- Hardware system features have a non-linear relationship with performance, hence the linear model performs poorly. Random Forests perform better than Decision Trees due to averaging error approach of the bagging method.

Our future work will include building efficient multivariate models to include power in addition to performance. and contribute more data sets to the with performance and power. We are also planning to apply transfer learning for performance prediction.

References

[1] P. Singleton, "Performance modelling - what, why, when and how," *BT Technology Journal*. [Online]. Available: https://doi.org/10.1023/A:1020860029447

[2] P. Malakar, P. Balaprakash, V. Vishwanath, V. Morozov, and K. Kumaran, "Benchmarking machine learning methods for performance modeling of scientific applications," in *2018 IEEE/ACM Performance Modeling, Benchmarking and Simulation of High Performance Computer Systems (PMBS)*, Nov 2018, pp. 33–44.

[3] G. Mariani, A. Anghel, R. Jongerius, and G. Dittmann, "Predicting cloud performance for hpc applications before deployment," *Future Generation Computer Systems*, vol. 87, pp. 618 – 628, 2018. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0167739X17312542

[4] A. Evangelidis, D. Parker, and R. Bahsoon, "Performance modelling and verification of cloud-based auto-scaling policies," *Future Generation Computer Systems*, vol. 87, pp. 629 – 638, 2018. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0167739X17312475

[5] J. Guo, A. Ma, Y. Yan, and B. Zhang, "Application performance prediction method based on cross-core performance interference on multi-core processor," *Microprocessors and Microsystems*, vol. 47, pp. 112 – 120, 2016. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0141933116300424

[6] J. L. Greathouse and G. H. Loh, "Machine learning for performance and power modeling of heterogeneous systems," in *2018 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, Nov 2018, pp. 1–6.

[7] J. Rico-Gallego, J. MartÃn, R. Manumachu, and A. Lastovetsky, "A survey of communication performance models for high-performance computing," *ACM Computing Surveys*, vol. 51, pp. 1–36, 02 2019.

[8] L. Lopez, M. Guynn, and M. Lu, "Predicting computer performance based on hardware configuration using multiple neural networks," in *2018 17th IEEE International Conference on Machine Learning and Applications (ICMLA)*, Dec 2018, pp. 824–827.

[9] S. K. Venkata, I. Ahn, D. Jeon, A. Gupta, C. Louie, S. Garcia, S. Belongie, and M. B. Taylor, "Sd-vbs: The san diego vision benchmark suite," in *2009 IEEE International Symposium on Workload Characterization (IISWC)*, Oct 2009, pp. 55–64.

[10] M. A. Heroux, D. W. Doerfler, P. S. Crozier, J. M. Willenbring, H. C. Edwards, A. Williams, M. Rajan, E. R. Keiter, H. K. Thornquist, and R. W. Numrich, "Improving Performance via Mini-applications," Sandia National Laboratories, Tech. Rep. SAND2009-5574, 2009.

[11] D. Bailey, E. Barszcz, J. Barton, D. Browning, R. Carter, L. Dagum, R. Fatoohi, P. Frederickson, T. Lasinski, R. Schreiber, H. Simon, V. Venkatakrishnan, and S. Weeratunga, "The nas parallel benchmarks," *Int. J. High Perform. Comput. Appl.*, vol. 5, no. 3, pp. 63–73, Sep. 1991. [Online]. Available: http://dx.doi.org/10.1177/109434209100500306

[12] M. Awad and R. Khanna, *Support Vector Regression*. Berkeley, CA: Apress, 2015, pp. 67–80. [Online]. Available: https://doi.org/10.1007/978-1-4302-5990-9_4

[13] A. E. Hoerl and R. W. Kennard, "Ridge regression: Biased estimation for nonorthogonal problems," *Technometrics*, vol. 12, no. 1, pp. 55–67, 1970. [Online]. Available: https://www.tandfonline.com/doi/abs/10.1080/00401706.1970.10488634

[14] C. Bishop, *Pattern Recognition and Machine Learning*, ser. Information Science and Statistics. Springer-Verlag New York, 2006.

[15] C. E. Rasmussen, *Gaussian Processes in Machine Learning*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2004, pp. 63–71. [Online]. Available: https://doi.org/10.1007/978-3-540-28650-9_4

[16] W.-Y. Loh, "Classification and regression trees," *WIREs Data Mining and Knowledge Discovery*, vol. 1, no. 1, pp. 14–23, 2011. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/widm.8

[17] L. Breiman, "Random Forests," *Machine Learning*, vol. 45, no. 1, pp. 5–32, 2001. [Online]. Available: http://link.springer.com/10.1023/A:1010933404324

[18] P. Geurts, D. Ernst, and L. Wehenkel, "Extremely randomized trees," *Machine Learning*, vol. 63, no. 1, pp. 3–42, Apr. 2006. [Online]. Available: http://link.springer.com/10.1007/s10994-006-6226-1

[19] J. H. Friedman, "Stochastic gradient boosting," *Computational Statistics & Data Analysis*, vol. 38, no. 4, pp. 367–378, Feb. 2002. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S0167947301000652

[20] T. Chen and C. Guestrin, "XGBoost: A Scalable Tree Boosting System," *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining - KDD '16*, pp. 785–794, 2016, arXiv: 1603.02754. [Online]. Available: http://arxiv.org/abs/1603.02754

[21] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.