Cross-Platform Performance Prediction with Transfer Learning using Machine Learning

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

***Abstract*—Machine-learning models are widely used for performance prediction due to its applications in the advancements of hardware-software co-development. Several researchers have focused on predicting the performance of an unknown target platform (or system) from the known performance of a particular platform (or system); we call this as the cross-platform prediction. Transfer learning is used to reuse previously gained knowledge on a similar task. In this paper, we use transfer learning for solving two problems cross-platform prediction and cross-systems prediction. Our result shows the prediction error of 15% in case of cross-systems (Simulated to Physical) prediction whereas in case of the cross-platform prediction error of 17% for simulation-based X86 to ARM prediction and 23% for physical Intel Core to Intel-Xeon system using best performing tree-based machine-learning model. We have also experimented with dimensionality reduction using PCA and selection of best hyper-parameters using grid search techniques.**

. ***Index Terms*—transfer learning; cross platform prediction; machine learning models; metrics;**

I. Introduction

Advancements in computer systems have helped us with the availability of several options of hardware configurations, each having different architecture features and performance. Predicting the performance of a system that we do not have access to is a challenging problem with the help of performance data of another system that we have access to. Cross-platform performance prediction is an active area of research that aims to predict the performance from one type of computer system (known platform) to the other (unknown target platform) with the help of transfer learning.

In transfer learning, machine-learning models are built with performance data of known platforms along with a small percentage of target platform data (that we have access to) to predict the performance data of the unknown target platform. In this work, we explore transfer learning to solve two problems; cross systems prediction and cross platform prediction. In cross systems prediction, we predict the performance of physical systems from performance collected from simulation-based systems while prediction of different target architecture is performed in cross platform prediction.

The remainder of the paper is organized as follows: Section II describes work related to transfer learning how our objective differs. Section III describes the selection of benchmarks for workload and computer hardware selection for simulation-based hardware and physical systems. Section IV details machine learning models along with techniques to improve prediction accuracy, such as grid search that we have used for the study. Section V provides results and analysis of cross-platform and cross-systems prediction, and evaluation of different machine-learning models including dimensionality reduction, hyper-parameter selection. Finally, section VI provides concluding remarks and work that we plan to continue.

II. Related Work

Transfer learning has been widely used in performance prediction research. Work in [1] predicts 90% of the performance data of server C using machine learning model trained on 100% features collected from server A and B and 10% from server C. [2] shows that a trained machine-learning model from one X86-based HPC system performance data can be re-trained using transfer learning on only one percent of samples from the test (another) X86-based HPC system to predict ninety-nine percent of performance data for the test system. A novel instance-based transfer learning technique is proposed in [3] for measuring performance of MariaDB database query responses installed on virtual machines with different system configuration and database size with incremental changes in system configuration or database size. Service-level metric prediction of data-center 2 or 3 is done using transfer learning from measurements collected from data-center 1 in [4]. Cross-platform prediction for ARM-based system from x86-based system in performed in [5].

The work in [1], [4] and [2] uses transfer learning for prediction of target systems having the same instruction-set (x86) while [5] perform cross-plaform prediction without transfer learning, In addition, to the best of our knowledge predicting performance of physical systems from simulated system (cross-system) is not addressed. Therefore, We address these questions: (i) Can we use transfer learning to predict the performance of ARM-based systems from X86-based performance? (ii) Can we use transfer learning to predict the performance of physical systems from simulated system's performance data? (iii) What is the effect of dimensionality reduction? (iv) How to to select hyper-parameter values of machine-learning models?

III. DataSet Preparation

A. Algorithms Selection for Workload

We have evaluated models described in section IV on several benchmark programs representative of real-world applications

TABLE I: Dataset information and type

Scientific Applications	System Type	Memory/Compute Bound (MB/CB)	Data Points
Dijkstra	Physical	MB	52
Dijkstra	Simulated	MB	475
Quick Sort	Physical	CB+MB	320
Quick Sort	Simulated	CB+MB	2850
Mantevo miniFE	Physical	CB	124
NPB EP	Physical	CB	108
NPB MG	Physical	MB	108
SHA	Physical	CB	52
SHA	Simulated	CB	475
Stitch	Physical	CB+MB	52
Stitch	Simulated	CB+MB	475
SVM	Physical	CB	52
SVM	Simulated	CB	475

having different computation and data access patterns listed in table I. Compute-bound (CB) applications SVM from SD-VBS [6], miniFE from the Mantevo benchmark suite [7], SHA from MiBench and EP algorithm from the NAS parallel benchmark (NPB) [8]. Memory-bound (MB) applications dijkstra from MiBench and MG algorithm from NPB. Similarly, we have used Quick Sort from MiBench, and Stitch from SD-VBS are dependent on both computation and data access making them CB+MB. These applications were executed on the simulated systems and physical systems from which the runtime was collected.

B. Computer Hardware Selection

We have selected computer systems that describe the class of computers commonly used today to execute the benchmark programs for collecting performance data. 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 the table II. We have collected the configuration details for these systems using 'dmidecode 'utility. For the simulation dataset, We have built 475 simulated-hardware system models using the Gem5 simulator, a widely accepted simulator for architectural research. To apply transfer learning, for cross-platform prediction on simulation data, we have chosen to build 120 hardware models based on the ARM instruction set (ISA) and 355 hardware models based on the X86 ISA. Gem5-simulated models have been built considering nine hardware model features, as shown in table III with values from the real memory and processor models commonly used in today's computers.

We have executed applications listed in the "Scientific Applications" column on each hardware type listed in the "System Type" column in table I. On physical systems, 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.

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

TABLE III: 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

IV. Models and Techniques Used

Supervised machine learning algorithms ease in empirical performance modeling. These models require training on an underlying dataset to learn the mapping function from input X_i to output Y_i. The machine learning models use the function or distribution learned during the training phase to predict output during the testing phase. Here, X_i can be a multidimensional tuple. In our case, X_i is the feature vector corresponding to processor and memory features as listed in tables II and III. And Y_i is the runtime or performance of an application concerning the underlying hardware feature vector X_i.

A. Machine Learning Models

TABLE IV: Machine Learning Models

Models Used	Abbr.	Important Hyper-Parameters
Support Vector Regr. [9]	svr	C, gamma, kernel
Linear Regr.	lr	fit_intercept, normalize
Ridge Regr. [10]	rr	alpha, fit_intercept, normalize, solver
K-Nearest Neighbor Regr. [11]	knn	p, n_neighbors, weights
Gaussian Process Regr. [12]	gpr	alpha, normalize_y, optimizer
Decision Tree Regr. [13]	dt	criterion, max_depth, max_features
Random Forest Regr. [14]	rf	n_estimators, criterion, max_depth
Extra Trees Regr. [15]	etr	n_estimators, criterion, max_depth
Gradient Boosting Regr. [16]	gbr	n_estimators, criterion, max_depth, loss
eXtreme Gradient Boosting [17]	xgb	n_estimators, max_depth, learning_rate
Deep Neural Network [18]	dnn	activation, loss, optimizer, metrics

Various models are listed in the ML literature. The diverse classes to which they relate are Regression models, Regularization, Bayesian, Decision Tree, Clustering, Ensemble, Dimensionality reduction, Instance-based and, Neural Network models. The most desirable model usually depends on dataset features like dimensionality, variance, skewness, sparsity, and

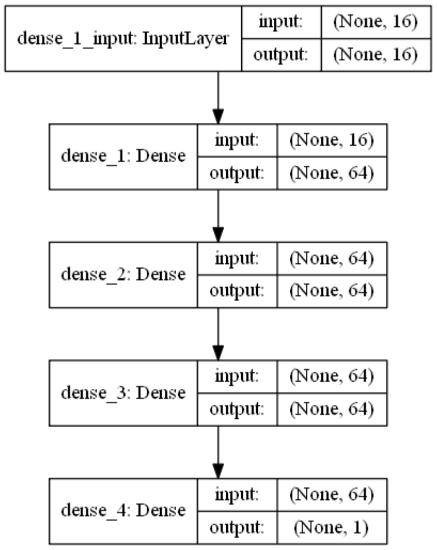


Fig. 1: DNN Model Summary

multivariate probability distribution. The models that we have used for the evaluation are listed in table IV. These models incorporate almost all types of classes that we have listed above. Furthermore, we have used Deep Neural Network (model summary, as shown in Fig.1. We used loss function as mean absolute error, optimizer as adam [19], and ran the model for 100 epochs in each case. ReLU activation is used for the first four layers and linear activation for the last layer.

B. Grid Search

Grid Search Technique [20] is used to fine-tune a model for best performance on specific data using cross-validation over a given parameter grid. The table IV lists the models with their respective hyper-parameters that were tuned for performing each experiment. The essential hyper-parameters which had a significant impact on model performance were used for the grid.

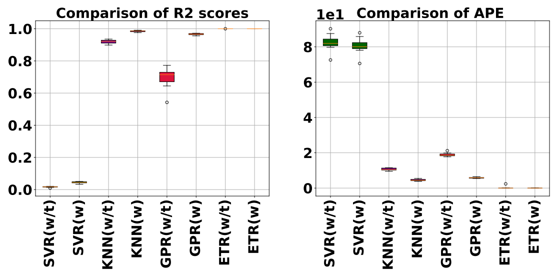
C. Transfer Learning

Transfer Learning aims to reuse previously gained knowledge on a specific task for application to a similar task. It is a tedious task to make different models for every different problem. To save time and computational cost, one can make a generalized model that can use learning from a previous problem and, after fine-tuning, performs well on other similar problems. For example, Alex-net architecture [21], is a generalized model for image classification and, after little fine-tuning, can be used for similar image classification tasks as used in [22] .

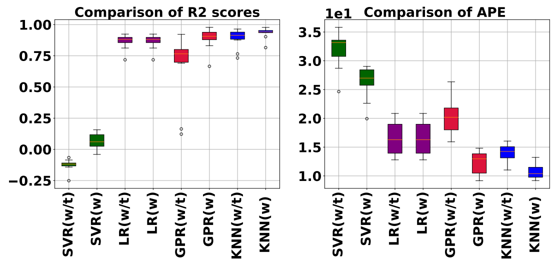
V. Experimental Evaluation

In this section, we performed several experiments to assess the prediction accuracy of machine learning models and the application of transfer learning for cross-prediction. We used categorical one-hot encoding to convert categorical data like instruction set architecture and memory type. Pre-processing also includes data normalization, and for that, we have used $StandardScaler()$ function from the sklearn library [20]. Furthermore, $ShuffleSplit()$ function [20], from sklearn library is used for cross-validation to ensure the robustness of the models. Please note that, for cross-prediction, we normalized both input and output data points, whereas, for other experiments, we just normalized the input.

The train to test ratio for each model is kept at 80:20 except for transfer learning experiments. The metrics used for model evaluation are R2 score, Median Absolute Percentage Error(MedAPE), and Mean Squared Error(MSE). Low values of MedAPE and MSE represent models that have high prediction accuracy, whereas R2 score of 1 means the model fits the data well. Negative R2 value reveals that the model fit is even worse than horizontal hyperplane. We have shown dataset characteristics and model evaluation in different contexts and scenarios. Firstly, the outcome of PCA (Principal Component Analysis) and Grid Search to increase model prediction accuracy is shown. Secondly, the prediction results of models for cross prediction using transfer learning are discussed. Thirdly, the analysis of performance prediction for each model, and lastly, comparison, according to various application types, is also conducted.



(a) Models with (w) and without (w/o) PCA on Quick Sort



(b) Models with (w) and without (w/o) Grid Search on Dijkstra

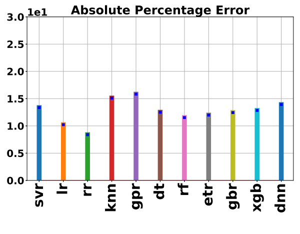
Fig. 2: Application of PCA and Grid Search Techniques

A. Effect of Dimensionality Reduction using PCA

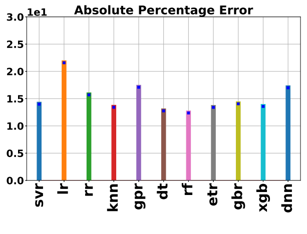
Principle Component Analysis (PCA) is a statistical method that identifies the correlated variables to transform a large number of input variables into uncorrelated variables with reduction. We have applied PCA to observe the effect of the dimensionality reduction on the performance prediction models. Fig. 2a illustrates the effect of PCA in improving performance prediction accuracy. The experiments are performed on the Quick Sort (Simulated) application dataset. As can be seen, models have lower error with Mean R2 = 0.75 and Mean MedAPE = 22.59% when PCA is applied in comparison to Mean R2 = 0.66 and Mean MedAPE = 27.77% without PCA. The results are averaged over all models (svr, knn, gpr, etr) as shown in Fig. 2a. Prediction accuracy improvement is due to machine-learning models' ability to predict better using a smaller number of uncorrelated variables rather than a larger number of variables having some amount of correlation. For example, in our experiment, it was found that most variance of our higher dimensional data is expressed within three to six principal components. Similar increase in model performance is also observed for other datasets when PCA is applied.

B. Effect of Grid Search on Model Performance

Grid search method helps us tune the hyper-parameters of machine learning algorithms that are not learned during the training phase. By performing an exhaustive search through the grid of parameters, we can select the hyper-parameters that provides the highest prediction accuracy from the model. Fig. 2b illustrates the effect of using Grid Search for improving the performance of machine learning models. The experiments are performed on Dijkstra (Simulated) application dataset. As can be seen, models have improved performance with Mean R2 = 0.69 and Mean MedAPE = 16.66% when Grid Search is used in comparison to Mean R2 = 0.57 and Mean MedAPE = 20.97% when default model arguments are used. These results are averaged over all models (svr, lr, gpr and knn) shown in Fig. 2b. The improvement in prediction accuracy using Grid Search is due to the selection of suitable values of hyperparameters such as regularization parameter 'alpha' to prevent model over-fitting, 'fit_intercept' parameter for bias in linear methods, 'kernel' function for feature transformation like 'rbf' in svm, stopping criterion like 'max_depth' parameter in tree-based algorithms, 'normalize' parameter for data normalization and, the 'metric' parameter for calculation of loss function. We tuned the parameters, as shown in table IV using an exhaustive grid search. In some cases, default values of hyper-parameters can be the best selection as observed in the case of linear regression (lr) the Fig. 2b.



(a) Dijkstra



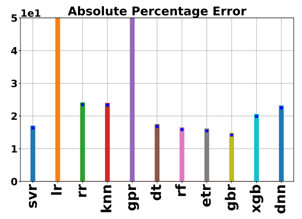
(b) SHA

Fig. 3: Results for Training on Simulated Data and Testing on Physical Data

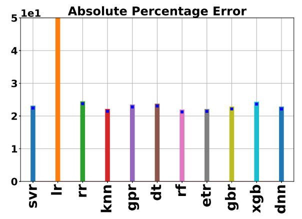
C. Cross Prediction using Transfer Learning

1) Cross Systems: Simulated to Physical Prediction

To predict the runtime for physical systems from simulation-based runtime, we implemented transfer learning on Dijkstra (memory-bound) and SHA (compute-bound) applications data to consider both types of applications (refer table I for dataset description). Firstly, for each application, we trained our models on 100% of Simulated Systems data and 10% of Physical Systems data. (Note: Since model.partial_fit() function for retraining is not available for all models in Sklearn library, we used model.fit() function and trained only once with combined data). After that, we predicted the performance of machine learning models on the rest 90% of Physical Systems data. For implementing transfer learning in dnn, we first trained the model (shown in Fig. 1) on 100% of Simulated Systems data. After that, we froze all model layers except the last 2 layers and again fine-tuned the model with 10% of Physical Systems data. And then predicted the performance of fine-tuned model on rest 90% of Physical Systems data. The results for the application of transfer learning are shown in Fig. 3a for Dijkstra and in Fig. 3b for SHA application data. First observation is among the two application types, memory-bound (Dijkstra) with mean APE = 12.56% gives better cross prediction accuracy than compute-bound (SHA) with mean APE = 14.99%. This is because the complex design of the processors has higher manufacturer variability causing larger runtime variations for compute-bound algorithms resulting in higher errors. Furthermore, we observe that linear regression models (lr and rr), perform best in the case of memory-bound application (as shown in Fig. 3a), whereas Bagging models (rf and etr) performs best in case of compute-bound application (refer Fig. 3b). Thus linear and bagging models are good for cross-system performance prediction.



(a) Qsort(Simulated)-X86 to ARM



(b) Qsort(Physical)-Intel Core to Intel Xeon

Fig. 4: Results for Training and Testing on Cross Platforms

2) Cross Platforms: X86-based systems to ARM-based systems Prediction

In this case, We use transfer learning to predict the runtime on ARM-based systems from X86-based systems on Quick Sort Simulated systems data since the data points are maximum (refer table I). We divided the dataset into two types of hardware architectures, namely X86 and ARM. Firstly, we trained our models on 100% data points of X86-based hardware models (1470 data points) and 1% of data points of ARM-based hardware models (7 data points). After that, we predicted the performance of machine learning models on the rest 99% (713 data points) of ARM hardware models. Also, we performed transfer learning with dnn (fine-tuning using 1% test data) in the same manner as explained in the above section. The results for the application of transfer learning are shown in Fig. 4a. We observed that tree-based models with a 16.43% mean percentage error give good prediction accuracy for cross-platform performance prediction from X86 to ARM architectures and outperform all other models, including deep neural networks in case of simulated systems.

3) Cross Platform: Intel Core to Intel Xeon

In this part, we used transfer learning to predict Intel-Xeon server systems runtime from Intel Core general-purpose systems runtime on Quick Sort Physical systems dataset(refer table I). We divided the dataset into two variants of hardware architectures, namely Intel Core and Intel-Xeon. Firstly, we trained our models on 100% data points of Intel Core-based hardware models (280 data points) and 10% of data points of Intel-Xeon hardware models (4 data points). After that, we predicted the performance of machine learning models on the rest 90% (36 data points) of Intel-Xeon hardware models. Also, we performed transfer learning with dnn in the same manner as explained in the above section. The results for the application of transfer learning are shown in Fig. 4b. We observed that tree-based models with a 22.30% mean percentage error, give good prediction accuracy for cross-platform performance prediction from Intel Core to Intel-Xeon architectures and outperform all other models including deep neural networks in case of physical systems.

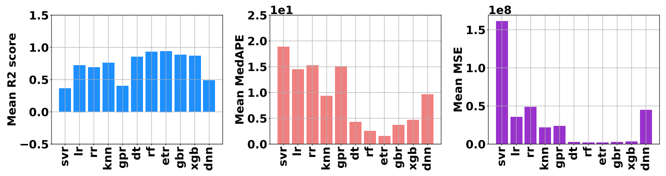


Fig. 5: Mean Prediction Scores per ML model across all datasets

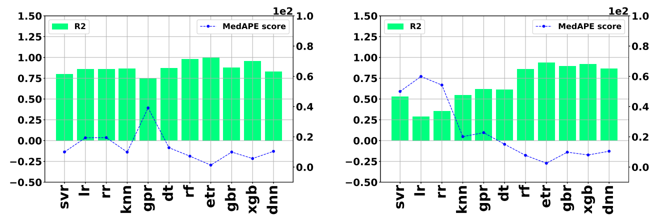
D. Model Comparison for Performance Prediction

Fig. 5 shows the value of R2 score, MedAPE and, MSE score for each model averaged over all datasets as listed in table I. We have observed that the models: 'etr', 'rf', 'gbr', 'dt', 'xgb', 'knn', 'lr', 'dnn', 'gpr', 'rr', 'svr' are in the order of decreasing prediction accuracy. This is observed after taking a majority vote of all three metrics named above. The inference that can be drawn from the above result and analysis for each model is as follows.

- ***svr***: Support Vector Regressor with mean R2 = 0.36 and mean MedAPE = 18.89% (Fig. 5), is underperforming as compared to other models. Grid Search found C (Regularization Parameter):'1000' and kernel: 'rbf' as the best parameters in almost all datasets. The higher value of C indicates that less regularization is required due to which a smaller margin hyper-plane is chosen.
- ***Linear Regression Models***: Linear Regressor with mean R2 = 0.72 and mean MedAPE = 14.49% and Ridge Regressor with mean R2 = 0.69 and mean MedAPE = 15.33% (Fig.5), both models are having nearly similar performance. Low prediction accuracy in linear models is due to the non-linear relationship between hardware features and runtime.
- ***knn***: KNN Regressor with mean R2 = 0.76 and mean MedAPE = 9.35% (Fig.5) gives better prediction accuracy than kernel svr and linear models. Using a grid search, we find the optimal number of neighbors (n_neighbors) and metric as 'Minkowski', which is a generalization of both euclidean and manhattan distance. The reason for better performance is that the KNN algorithm is robust to outliers, whereas svr and lr are not.
- ***gpr***: Gaussian Process Regressor with mean R2 = 0.40 and mean MedAPE = 15.11% (Fig.5) still performs better than ridge regressor and kernel svr. The model underperforms in comparison to other models because our data may not exactly follow the multivariate gaussian distribution. So, while learning posterior distribution by gpr, it may not be not tractable by the model.
- ***dt***: Decision Tree Regressor with mean R2 = 0.85 and mean MedAPE = 4.26% (Fig.5) gives better prediction accuracy than all the models discussed above. The parameters 'criterion' and 'max_depth' are optimized using Grid Search. The improved performance is due to the optimal splitting criterion at each node during tree formation. The decision tree finds training instances that follow the same decision rules as the test examples rather than considering correlation like other linear models.
- ***Bagging Models*** : rf and etr are the two bagging approach models that are outperforming all machine learning models due to the creation of a large number of decision trees. Extra Trees Regressor with mean R2 = 0.94 and mean MedAPE = 1.58% (Fig.5) performs better than Random Forest Regressor with mean R2 = 0.93 and mean MedAPE = 2.56%. The reason is due to random splits in Extra Trees, which works better in case features are noisy.
- ***Boosting Models***: Gradient Boosting Regressor with mean R2 = 0.88 and mean MedAPE = 3.73% (Fig.5) performs better than eXtreme Gradient Boosting with mean R2 = 0.86 and mean MedAPE = 4.72%. These boosting models work on bias reduction to get better prediction accuracy and sometimes lead to overfitting. Hence, the error in boosting models is higher than the bagging approach.
- ***dnn***: Deep Neural Network model (as shown in Fig.1) with mean R2 = 0.48 and mean MedAPE = 9.64% (Fig.5) is not able to outperform other machine learning models except svr and gpr. The reason for this performance is due to the less training dataset size. DNNs require a sufficient number of data points for learning the mapping function.

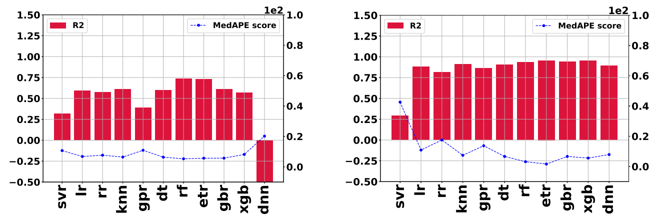
E. Estimating the effect of performance with respect to Compute Bound, Memory Bound and Both type of Applications

Figures 6, 7 and 8 shows R2 and MedAPE scores on left and right y-axis with ranges $[-0.5, 1.5]$ and $[-10\%, 100\%]$ respectively for compute-bound, memory-bound and both bound application types from dataset (refer table I).



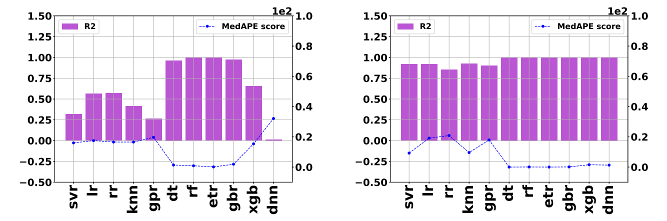
(a) Mantevo-MiniFE-Physical (b) NPB-EP-Physical

Fig. 6: R2 score and MedAPE for Compute-Bound Applications



(a) Dijkstra-Physical (b) NPB_MG-Physical

Fig. 7: R2 score and MedAPE for Memory-Bound Applications



(a) QuickSort-Physical (b) QuickSort-Simulated

Fig. 8: R2 score and MedAPE for Both Memory and Compute-Bound Applications

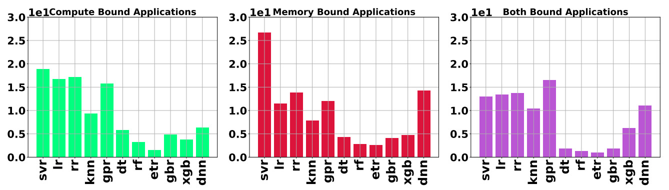


Fig. 9: Mean MedAPE for all application types

(Note: Here we have used color codes for each application type). The results are compiled for each dataset but we have shown performance of only two datasets per application type. Furthermore, Fig. 9 shows the mean MedAPE score for all three application types with scores averaged for all datasets (as listed in table I) for each machine learning model. The first observation from all these figures is that five models, which are 'dt', 'etr', 'rf', 'gbr' and, 'xgb' are performing well in nearly all scientific applications. This is also shown in the above evaluation section D. These models provide better prediction accuracy for datasets listed in table I. So for the next observations, we have considered only the result of these five tree-based models for comparison of overall performance.

The second observation that can be taken from Fig.9 is that on an average taken over MedAPE values of the tree-based models, prediction accuracy in Both Memory and Compute-Bound applications (with 7.92% mean error) is the highest followed by Memory-Bound applications (with 9.06% mean error) and lowest accuracy in the Compute-Bound applications (with 9.70% mean error). This observation strengthens the conclusion from previous section C (part 1) that compute-bound applications have more significant runtime variations due to the sophisticated design of the processors resulting in higher errors. The memory-bound applications have a higher dependence on memory features, which hides the effect manufacturer variability of processor, causing fewer runtime variations and lower error.

VI. Conclusion And Future Work

We have shown in this work that using transfer learning, cross-prediction in terms of simulation-based systems to physical systems and prediction between systems with different instruction set can be achieved with good prediction accuracy. The experimentation consists of thirteen different databases (refer table I) collected by executing tests of five scientific applications on 475 simulated models made in the Gem5 simulator, while eight scientific applications were executed on eight different physical computer systems. The conclusion that can be drawn from the above experiments are as follows:

- Dimensionality reduction using PCA is an important pre-processing step and increases model accuracy.
- Grid Search proved useful for optimal parameters selection and achieving better prediction accuracy.
- Cross-systems performance prediction from simulation to physical systems using transfer learning yields less than 15% percentage errors using tree-based models.
- Cross-platform performance prediction from Intel to ARM architectures on simulation systems data set using transfer learning yields less than 17% percentage errors using tree-based models.
- Cross-platform performance prediction in physical systems data set from Intel Core to Intel-Xeon architectures using transfer learning yields less than 23% percentage errors using tree-based models.
- Performance prediction on simulated to simulated or physical to physical systems, the tree-based machine learning models, namely etr, rf, gbr, dt and, xgb outperform all other machine learning models. Among these models, bagging-based models perform better.
- Due to variability in processor manufacturing, prediction accuracy in memory-bound applications is higher than compute-bound applications.

In future, we plan to include experiments with multi-target performance and power prediction using transfer learning. We will contribute datasets with performance and power for different hardware models to the research community.

References

[1] J. Sun, G. Sun, S. Zhan, J. Zhang, and Y. Chen, "Automated Performance Modeling of HPC Applications Using Machine Learning," *IEEE Transactions on Computers*, vol. 69, no. 5, pp. 749–763, may 2020.

[2] P. Malakar, P. Balaprakash, V. Vishwanath, V. Morozov, and K. Kumaran, "Benchmarking machine learning methods for performance modeling of scientific applications," in *Proceedings of PMBS 2018: Performance Modeling, Benchmarking and Simulation of High Performance Computer Systems, Held in conjunction with SC 2018: The International Conference for High Performance Computing, Networking, Storage and Analysis*. Institute of Electrical and Electronics Engineers Inc., feb 2019, pp. 33–44.

[3] F. Iorio, A. B. Hashemi, M. Tao, and C. Amza, "Transfer learning for cross-model regression in performance modeling for the cloud," in *Proceedings of the International Conference on Cloud Computing Technology and Science, CloudCom*, vol. 2019-Decem. Iorio2019: IEEE Computer Society, dec 2019, pp. 9–18.

[4] F. Moradi, R. Stadleryz, and A. Johnsson, "Performance Prediction in Dynamic Clouds using Transfer Learning," in *2019 IFIP/IEEE Symposium on Integrated Network and Service Management (IM).*, 2019.

[5] X. Zheng, L. K. John, and A. Gerstlauer, "LACross: Learning-Based Analytical Cross-Platform Performance and Power Prediction," *International Journal of Parallel Programming*, vol. 45, no. 6, pp. 1488–1514, dec 2017.

[6] S. Venkata, I. Ahn, D. Jeon, A. Gupta, C. Louie, S. Garcia, S. Belongie, and M. Taylor, "SD-VBS: The San Diego Vision Benchmark Suite," in *Proceedings of the 2009 IEEE International Symposium on Workload Characterization, IISWC : October 4-6 2009, Austin, TX, USA*. IEEE, 2009.

[7] P. Crozier, H. Thornquist, R. Numrich, A. Williams, H. Edwards, E. Keiter, M. Rajan, J. Willenbring, D. Doerfler, and M. Heroux, "Improving performance via mini-applications," 01 2009.

[8] D. H. Bailey, E. Barszcz, J. T. Barton, D. S. Browning, R. L. Carter, L. Dagum, R. A. Fatoohi, P. O. Frederickson, T. A. Lasinski, R. S. Schreiber, H. D. Simon, V. Venkatakrishnan, and S. K. Weeratunga, "The nas parallel benchmarks," *International Journal of High Performance Computing Applications*, vol. 5, no. 3, pp. 63–73, 1991.

[9] 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

[10] 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

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

[12] C. E. Rasmussen, "Gaussian processes in machine learning," in *Summer School on Machine Learning*. Springer, 2003, pp. 63–71.

[13] 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

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

[15] 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

[16] 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

[17] 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. [Online]. Available: http://arxiv.org/abs/1603.02754

[18] H. Wang and B. Raj, "On the Origin of Deep Learning," apr 2017.

[19] D. P. Kingma and J. Ba, "Adam: A Method for Stochastic Optimization," dec 2014. [Online]. Available: http://arxiv.org/abs/1412.6980

[20] F. Pedregosa FABIANPEDREGOSA, V. Michel, O. Grisel OLIVIERGRISEL, M. Blondel, P. Prettenhofer, R. Weiss, J. Vanderplas, D. Cournapeau, F. Pedregosa, G. Varoquaux, A. Gramfort, B. Thirion, O. Grisel, V. Dubourg, A. Passos, M. Brucher, M. Perrot andÉdouardand, andÉdouard Duchesnay, and F. Duchesnay EDOUARDDUCHESNAY, "Scikit-learn: Machine Learning in Python Gaël Varoquaux Bertrand Thirion Vincent Dubourg Alexandre Passos PEDREGOSA, VAROQUAUX, GRAMFORT ET AL. Matthieu Perrot," Tech. Rep., 2011. [Online]. Available: http://scikit-learn.sourceforge.net.

[21] A. Krizhevsky, I. Sutskever, and G. Hinton, "ImageNet Classification with Deep Convolutional Neural Networks," *Neural Information Processing Systems*, vol. 25, apr 2012.

[22] A. Saini, T. Gupta, R. Kumar, A. K. Gupta, M. Panwar, and A. Mittal, "Image based indian monument recognition using convoluted neural networks," in *2017 International Conference on Big Data, IoT and Data Science (BID)*, 2017, pp. 138–142.