Performance and Power Prediction on Disparate Computer Systems

by

Aditya Amrutiya
201811028

A Project Report Submitted in Partial Fulfilment of the Requirements for the Degree of

MASTER OF TECHNOLOGY
in
INFORMATION AND COMMUNICATION TECHNOLOGY
to

DHIRUBHAI AMBANI INSTITUTE OF INFORMATION AND COMMUNICATION TECHNOLOGY

May, 2020

Declaration

I hereby declare that

i) the project comprises of my original work towards the degree of Master of Technology in Information and Communication Technology at Dhirubhai Ambani Institute of Information and Communication Technology and has not been submitted elsewhere for a degree,

ii) due acknowledgment has been made in the text to all the reference material used.

Aditya Amrutiya

Certificate

This is to certify that the project work entitled Performance and Power Prediction on Disparate Computer Systems has been carried out by Aditya Amrutiya for the degree of Master of Technology in Information and Communication Technology at *Dhirubhai Ambani Institute of Information and Communication Technology* under my/our supervision.

Prof. Amit Mankodi
Project Supervisor

Acknowledgments

I want to acknowledge the help of my colleague Rajat Kumar for carrying out experimentation, literature review, and paper writing with me. I also want to thanks our Project Supervisor, Prof. Amit Mankodi, for the project idea, consistent motivation, review, and updates on the project from time to time with hard deadlines. Moreover, I want to thanks Prof. Amit Bhatt and Prof. Bhaskar Chaudhary for paper reviews and constant supervision. Because of this, we were able to submit three research articles in different conferences, as briefed in chapter 1.

I am also thankful to the DA-IICT administration for providing state of the art facilities and information updates to carry out research. Even amid the Covid-19 outbreak, there was constant motivation and support for completing the project. I highly appreciate the efforts of my colleagues, supervisors, and college administration.

Contents

Abstract

Performance and Power 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 evaluating the performance of machine learning models. 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).Our results for performance prediction show that the tree-based machine-learning models outperform all other models with median absolute percentage error (MedAPE) of less than 5% consisting of bagging and boosting models that help to improve weak learners. We have also observed that in physical systems, the prediction accuracy of memory-bound applications is higher as compared to compute-bound algorithms due to manufacturer variability in processors.
Moreover, the prediction accuracy is higher on simulation-based hardware due to its deterministic nature as compared to physical systems. We have used 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 prediction whereas in case of the cross-platform prediction error of 17% for simulation-based X86 to ARM system using best performing tree-based machine-learning model. For the prediction of the power consumption along with that of performance we have employed several machines learning univariate or multivariate models in our experiments. Our result shows that runtime and power prediction accuracy of more than 80% and 90% respectively is achieved for multivariate deep neural network model in cross-platform prediction. Similarly, for cross-system prediction runtime accuracy of 90% and power accuracy of 75% is achieved for the multivariate deep neural network.

Keywords: *Performance and Power Prediction, Machine Learning, Transfer Learning, Multivariate Prediction, Cross-Prediction*

List of Tables

List of Figures

CHAPTER 1
Submission Details

Thanks to the dedication of co-author: Rajat Kumar and guide: Prof. Amit Mankodi, we were able to submit the following papers to conferences as listed below within deadline:

1. **Paper 1: Evaluating Machine Learning Models for Disparate Computer Systems Performance Prediction [1]**
   Conference : IEEE CONECCT 2020: International Conference on Electronics, Computing and Communication Technologies
   Location : Bangalore, India
   Notification of Acceptance : May 15, 2020

2. **Paper 2: Cross-Platform Performance Prediction with Transfer Learning using Machine Learning [31]**
   Conference : 11th ICCCNT 2020: 11th International Conference on Computing Communication and Networking Technologies
   Location : IIT Kharagpur, West Bengal, India
   Notification of Acceptance : May 30, 2020

3. **Paper 3: Modeling Performance and Power on Disparate Platforms using Transfer Learning with Machine Learning Models [2]**
   Conference : CoMSO2020: International Conference on Modeling, Simulations and Optimizations
   Location : NIT Silchar, India
   Notification of Acceptance : May 25, 2020

Also a special thanks to Prof. Amit Bhatt and Prof. Bhaskar Chaudhary for reviewing the paper.

CHAPTER 2
Introduction

Performance and Power have been two crucial factors that are considered while selecting a computer system. The performance or runtime used to be the primary criteria for selection in the past, but the power consumption of computing resources has emerged as a significant concern in recent times due to environmental hazards. Advancements in computer systems have helped us with the availability of several options of hardware configurations, each having different architecture features and performance. It is useful for software developers and system architects to choose the most optimal strategy of application and configuration of the underlying hardware systems. The estimation of Runtime and Power of an application is determined by feature of underlying hardware on which it is executed such as processor and memory features. The performance of compute bound applications will depend more on processor clock speed, number of cores and cache, etc. On the other hand, memory bound applications will depend more on memory clock, type, etc. Furthermore, predicting the performance of a system (that we do not have access) 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. Therefore, performance and power prediction models using machine learning have become an active area of research.

The study in [26] 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 [28] deals with predicting optimal cloud configurations for HPC before deployment.The work in [15] 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 [14]. 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. The prediction accuracy rate of 97.5% was achieved in [24] using multiple neural networks and PCA on SPEC CPU2006 and SPEC CPU2017 benchmarks.

Several researchers have used power prediction to reduce power consumption in computer systems. Work in [20] uses power consumption obtained through prediction as feedback to improve power supply units used in physical computer systems. In [22] and [34] system-level power prediction is performed to understand effect of component-level usage on power consumption. By enabling efficient thread-level parallelism in heterogeneous cores, power consumption is reduced in [25]. Cross-platform prediction aid us in predicting performance and power from one platform or system to the other. Software performance and power on the ARM-based hardware system is predicted in [37] by collecting performance counters on X86 systems using linear regression univariate model. Work in [27] shows that a trained machine learning model from one HPC system can be re-trained on only one percent of samples from test (another) HPC system to predict ninety-nine percent of performance data. In this work, we have explored the transfer learning technique for cross-platform or cross-system prediction.

Authors of [19] have used transfer learning for improving the configuration of highly configurable software in real systems using a small set of input from the simulation system. Transfer learning is used in [29] to fine-tune the neural network model by re-training it in cloud services with incremental data, thus saving extra time to re-train the model from scratch. The GPU performance is predicted using features that affect it's performance time, collected during the execution of an algorithm on the x86-based system in [6] and [3]. For an unseen software, performance on ARM-based hardware system is predicted in [37] by collecting performance counters on X86 systems.

We have identified questions from related work, which are (for both univariate and multivariate case): (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? (iv) Can we predict the performance of ARM-based systems from X86-based performance data using transfer learning? (v) Can we predict the performance of physical sys-

tems using a simulation-based system's performance data using transfer learning for both cases? (vi) What is the effect of dimensionality reduction on performance prediction? (vii) How to effectively select values of hyper-parameters for machine-learning models? We address all these questions with our experimentation in the evaluation section.

CHAPTER 3
Dataset Preparation

3.1 DataSet Preparation

3.1.1 Algorithms Selection for Workload

Table 3.1: Dataset Information and Type for All Applications

Scientific Applications	System Type	Data Point	System Type	Data Point	Memory/Compute Bound (MB/CB)
Dijkstra [16]	Physical	52	Simulated	362	MB
Matrix Mult. [35]	Physical	519	Simulated	1780	MB
Montecarlocalcpi[35]	Physical	260	Simulated	1365	CB
MSER [35]	Physical	52	Simulated	430	CB
Quick Sort [16]	Physical	672	Simulated	2730	MB+CB
SHA [16]	Physical	52	Simulated	367	CB
Stitch [35]	Physical	52	Simulated	425	MB+CB
SVM [35]	Physical	52	Simulated	390	CB
Tracking [35]	Physical	52	Simulated	425	MB
Mantevo miniFE [11]	Physical	124	-	-	CB
NPB EP [5]	Physical	108	-	-	CB
NPB MG [5]	Physical	108	-	-	MB

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[35], miniFE[17] from the Mantevo Benchmark Suite [11], and EP algorithm from the NAS parallel benchmark[5]. The memory-bound application consists of Matrix Multiplication, Tracking algorithm from SD-VBS[35], and MG algorithm from the NAS benchmark[5]. These applications were executed on the simulated environment and physical systems from which the runtime and power was collected. Table 3.2 shows the scientific applications which are used for each paper.

Table 3.2: Use of Scientific Applications in Each Paper

Scientific Applications	Used in Paper
Dijkstra	1,3
Matrix Mult	1,3
Montecarlocalcpi	1,3
MSER	1,3
Quick Sort	2,3
SHA	2,3
Stitch	2,3
SVM	2,3
Tracking	1,3
Mantevo MiniFE	1,2
NPB EP	1,2
NPB MG	1,2

Table 3.3: 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

3.1.2 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 3.4. 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 3.3 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 3.1. For each algorithm, we have exe-

Table 3.4: 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

cuted 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.

CHAPTER 4
Models and Techniques Used

4.1 Machine Learning Models

The machine learning models are of classification and regression type and are classified into unsupervised and supervised category. According to the need of dataset we have for evaluation we have used the regression based supervised machine learning algorithms to performance modeling using empirical evaluation. 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 (runtime) in Paper 1[1] and Paper 2 [31] and multivariate (runtime and power) in Paper 3 [2]. So, the different ML algorithms are used to derive that function $f_{(x)}$ which gives the most accurate predicted Y value or values. In our case, each sample X_i is a multidimensional tuple corresponding to processor and memory characteristics as shown in table 3.4 for physical systems and table 3.3 for simulated systems. The overview of the models that we use in our paper is as follows.

4.1.1 Support Vector Regressor (svr)

Support Vector Regression [4] 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.

4.1.2 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:

$$y = \alpha + x\beta + \epsilon \tag{4.1}$$

and a multi-variable or multiple linear regression model takes the form:

$$y = \alpha + x_1\beta_1 + x_2\beta_2 + ... + x_k\beta_k + \epsilon \tag{4.2}$$

,where y is a dependent variable, x is a single value in the simple regression model, and $x_i's$ are the multiple values in the multiple regression model.

4.1.3 Ridge Regression (rr)

Ridge regression [18] is an extension for linear regression when a penalty term is added to the loss function to avoid over-fitting and when there is multi collinearity among the features.

4.1.4 K Nearest Neighbors (knn)

The class of instance-based methods has K Nearest Neighbors Regression[8]. The KNN tries to predict the output based on the distance between the neighboring instances. The model does not require a prior training it just finds the k neighboring instances of test sample and averages the target value of those k neighbors to provide us the target value for our test sample.

4.1.5 Gaussian Process Regressor (gpr)

Gaussian process regression[32] model tries to solve regression problem using gaussian processes. It has an equation $Y(x) = \sum_{i=1}^{n} \beta_i f_i(x) + Z(x)$ [33] where $Z()$ is a Gaussian process, $f(x)$'s are known functions, β is unknown and Y is response that we are interested in. Similar to linear regression models gpr also can output two target values directly but internally (in Sklearn library) it has a separate model created for each target value.

4.1.6 Decision Trees Regressor (dt)

Decision Tree regression[23] 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.

4.1.7 Random Forest Regressor (rf)

Random Forest[9] 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.

4.1.8 Extremely Randomized Trees (etr)

Extremely randomized trees [13] 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.

4.1.9 Gradient Boosting Regressor (gbr)

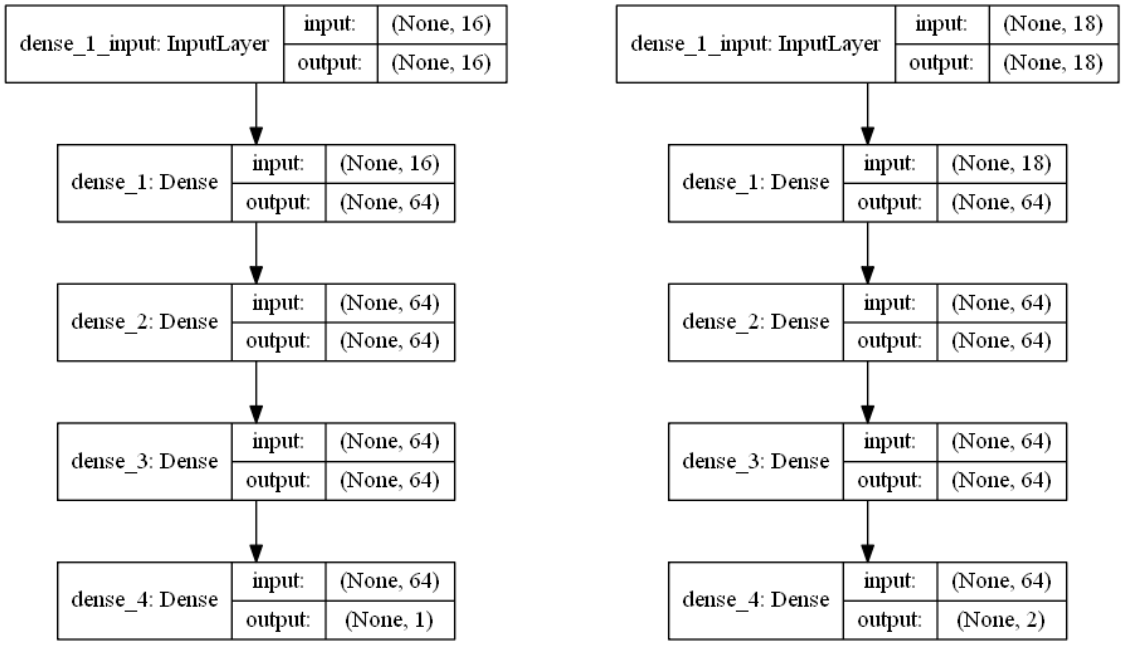
Gradient boosting regression [12] is a boosting technique where weak learners are combined to get a strong learner. Decision Trees are used as weak learners. Trees are formed in a greedy manner while choosing the best split using information gain. Trees follow an additive approach one at a time. Gradient 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.

4.1.10 XGBoost (xgb)

XGBoost[10] 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.

4.1.11 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 4.1 for Paper 1 [1], 4.1a for Paper 2 [31] and, 4.1b for Paper 3 [2]. 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 dnn model except for transfer learning experiments.



(a) DNN Summary used in Paper 2 [31]

(b) DNN Summary used in Paper 3 [2]

Figure 4.1: DNN Models Summary

Table 4.1: Model Summary of DNN Variants used in Paper 1 [1]

	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

4.2 Grid Search

A single machine learning model has a number of parameters which are to be set before the learning process begins. Grid Search Technique [30] is used to fine-tune a model for best performance on specific data using cross-validation over a given parameter grid. The essential hyper-parameters which had a significant impact on model performance were used for the grid.

4.3 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.

CHAPTER 5
Experimental Evaluation

5.1 Evaluation Process

For finding the accuracy of the runtime / power prediction system we carried out the following steps.

- First of all the csv file of an algorithm executed on multiple physical and simulated environment is collected containing processor and memory features of system along with the actual runtime and power consumption data.
- The rows with any NULL entries are removed from the dataset.
- Categorical data from 'isa' and 'mem-type' attributes is converted to the real valued entity using one-hot encoding scheme.
- $StandardScaler()$ function from sklearn library[30] is used to normalize the data.
- Then we used Grid Search for each of the machine learning algorithm mentioned in 4.1 to find the best hyper-parameters for training
- Machine learning models are then trained on the best parameters with train test ratio 80:20 in most of the cases and 10 fold cross validation is carried out using $ShuffleSplit()$ function [30] to assess the robust nature of the models.
- The resultant R2 score , Median and Mean Absolute Percentage Error and Maximum Squared Error is noted in a resultant csv for further analysis.

5.2 Results from Paper : 1

For doing the analysis of performance or runtime we have performed experiments on the dataset mentioned in table 3.2. We have applied machine learning models described in section 4.1 to each of the dataset. We have evaluated machine learning models in different scenarios. Firstly, the overall result for each model averaged over all nine datasets and, finally, a comparison of model performance between physical and simulated systems data is made.

5.2.1 Model comparison for performance prediction

Fig. 5.1 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.5.1 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. 5.1), 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.5.1), 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. 5.1 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.5.1) 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.5.1) 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.5.1) 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.

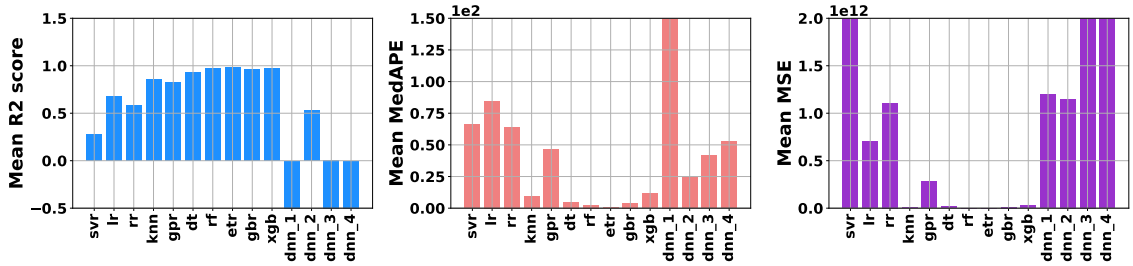


Figure 5.1: Mean of R2, MedAPE and MSE values per model for all datsets

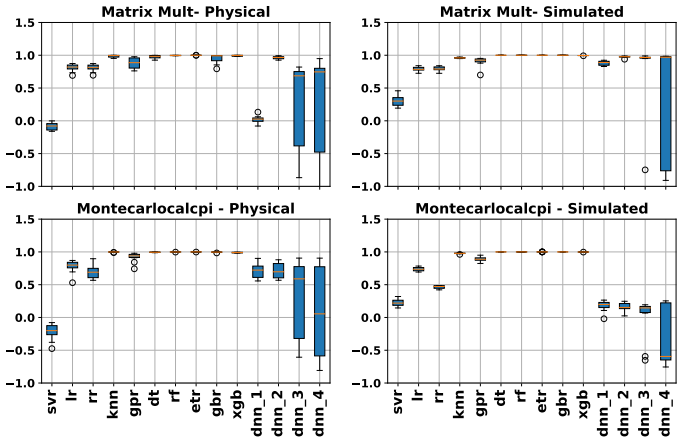
- ***rf and etr***: These two models are giving best performance prediction accuracy. Both algorithms are bagging approaches of the decision tree, so a large number of trees improve the accuracy of the model. Extra Trees Regressor with mean R2 = 0.99 and mean MedAPE = 0.69% (Fig.5.1) performs better 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.5.1) 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.

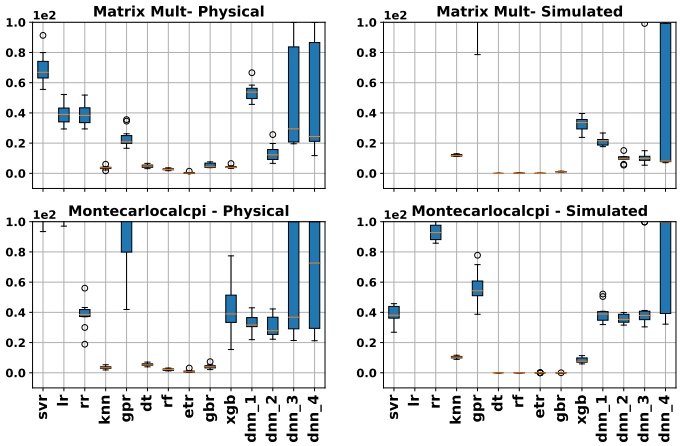
- ***dnn***: Deep Neural Networks (with best model among the four variants (table 4.1) to be dnn_2) with mean R2 = 0.53 and mean MedAPE = 23.98% (Fig.5.1) are not able to perform well here. We believe that due to limited number of data points available (as can be seen in our case in table 3.1), neural network models have lower accuracy than tree-based models.

5.2.2 Comparing model performance on Physical and Simulated Systems

Fig. 5.2 shows R2 score and MedAPE gathered from 10-fold cross-validation for performance prediction on physical and simulated systems for Matrix Multiplication and Motecarlocalcpi 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



(a) R2 score



(b) Absolute Percentage Error

Figure 5.2: Comparing model performance on Physical and Simulated Systems

= 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.

5.2.3 Comparing model performance on Memory/Compute Bound Systems

From figure 5.2 we 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 applica-

tions perform better. In the case of simulated systems, the Gem5 simulator [7], 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.

5.3 Results from Paper : 2

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.

5.3.1 Effect of Dimensionality Reduction using PCA

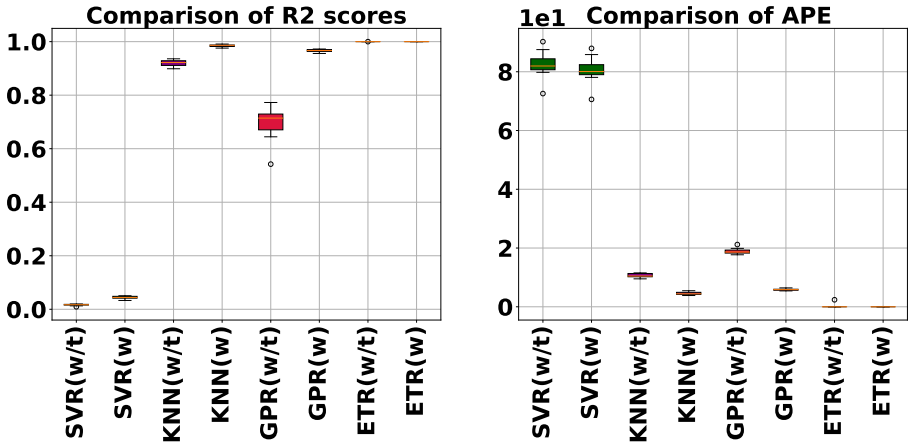


Figure 5.3: Models with (w) and without (w/o) PCA on Quick Sort Application

Principle Component Analysis (PCA) is a statistical method that is used to convert a large number of co-related input variables to a small number of uncorelated input variables. We have applied PCA to observe the effect of the dimensionality reduction on the performance prediction models. PCA keeps only those input variables that have the largest impact on the output value. Fig. 5.3 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. 5.3. 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. Similar increase in model performance is also observed for other datasets when PCA is applied.

5.3.2 Effect of Grid Search on Model Performance

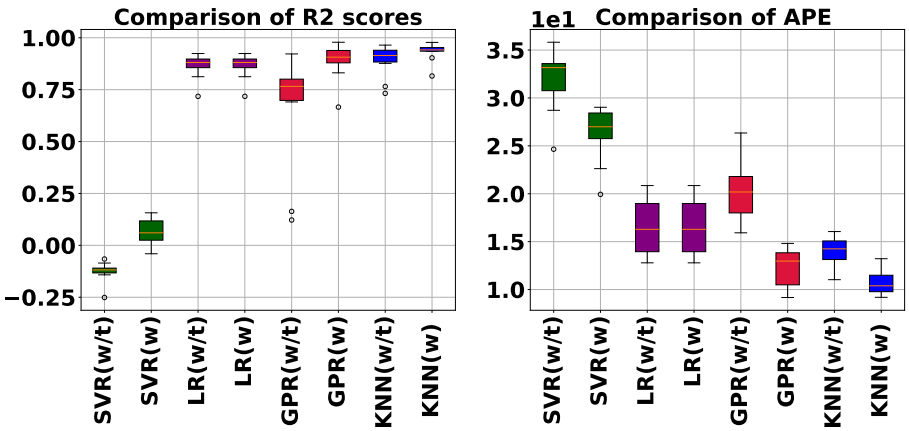
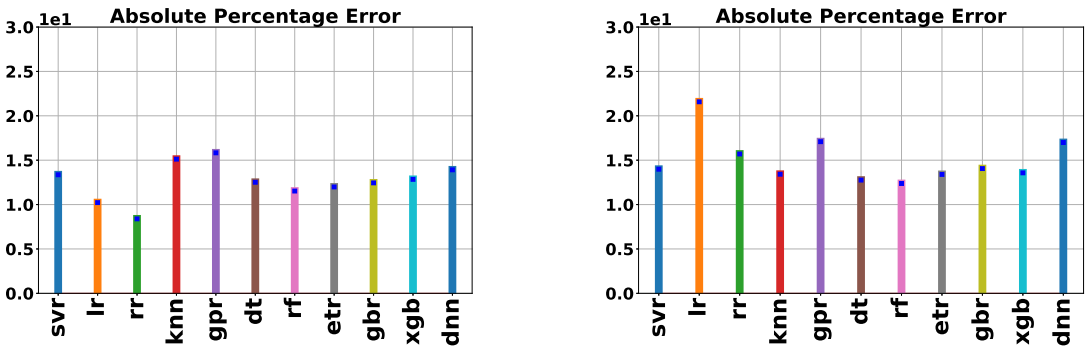


Figure 5.4: Models with (w) and without (w/o) Grid Search on Dijkstra Application

A model hyperparameter is a characteristic of a model that is external to the model and whose value cannot be estimated from data. The value of the hyperparameter has to be set before the learning process begins. Grid-search is used to find the optimal hyperparameters of a model which results in the most 'accurate' predictions. Fig. 5.4 illustrates the effect of using Grid Search. 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. 5.4. 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. We tuned the parameters 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) in the Fig. 5.4.

5.3.3 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 3.1 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 retrain-



(a) Dijkstra

(b) SHA

Figure 5.5: Results for Training on Simulated Data and Testing on Physical Data

ing 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. The results for the application of transfer learning are shown in Fig. 5.5a for Dijkstra and in Fig. 5.5b 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. 5.5a), whereas Bagging models (rf and etr) performs best in case of compute-bound application (refer Fig. 5.5b). Thus linear and bagging models are good for cross-system performance prediction.

2. **Cross Platform: 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 3.1). 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 5.6. We observed that tree-based models with a 16.43% mean percentage error give good prediction accu-

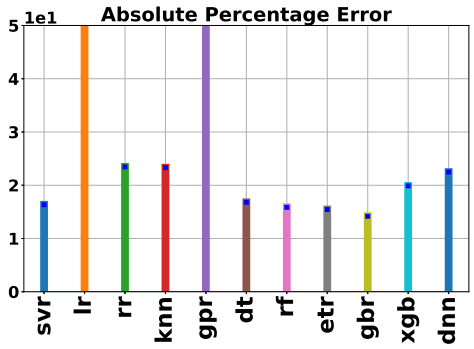
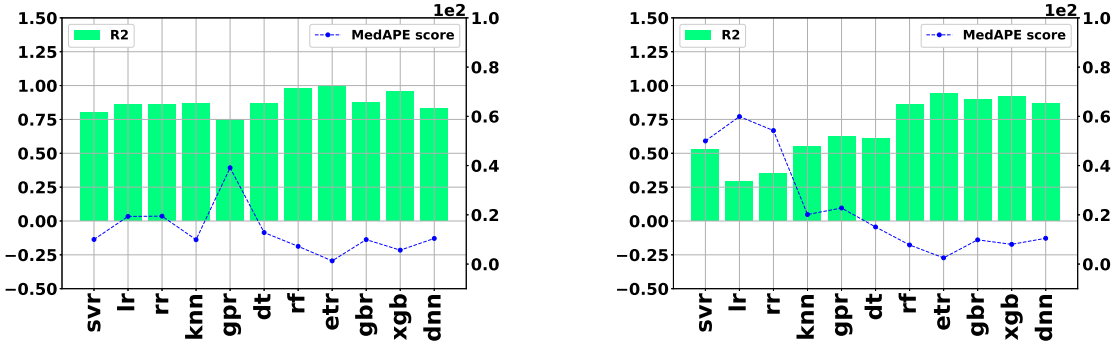


Figure 5.6: Qsort (Simulated) -X86 to ARM

racy for cross-platform performance prediction from X86 to ARM architectures and outperform all other models, including deep neural networks in case of simulated systems.

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

Figures 5.7, 5.8 and 5.9 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 3.1).



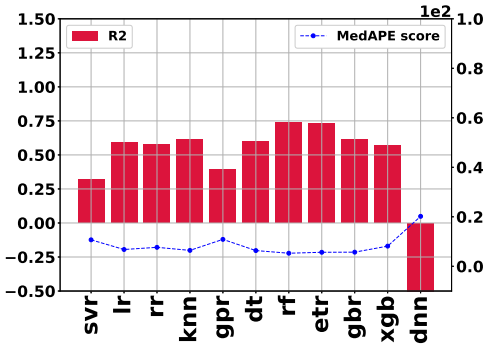
(a) Mantevo-MiniFE-Physical

(b) NPB-EP-Physical

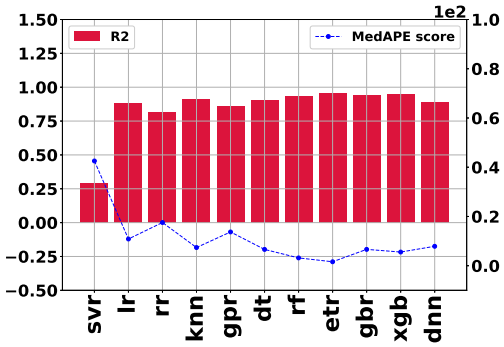
Figure 5.7: R2 score and MedAPE for Compute-Bound Applications

(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. 5.10 shows the mean MedAPE score for all three application types with scores averaged for all datasets (as listed in table 3.2) for each machine learning model.

The first observation from all these figures is that five models, which are 'dt', 'etr', 'rf',

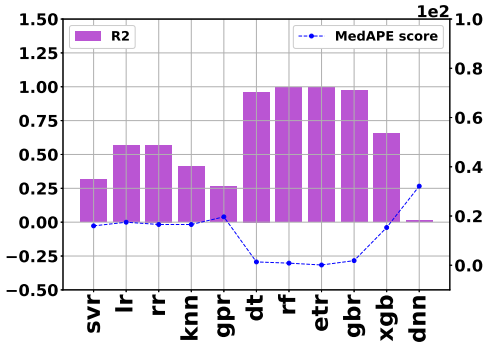


(a) Dijkstra-Physical

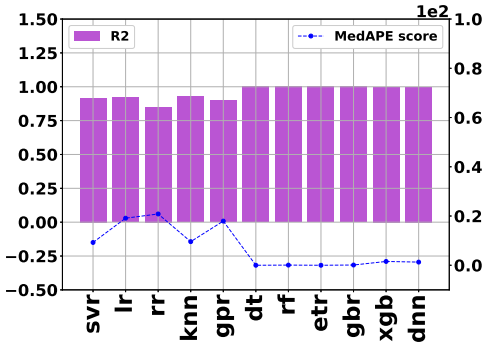


(b) NPB_MG-Physical

Figure 5.8: R2 score and MedAPE for Memory-Bound Applications



(a) QuickSort-Physical



(b) QuickSort-Simulated

Figure 5.9: R2 score and MedAPE for Both Memory and Compute-Bound Applications

'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 3.2. 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.5.10 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.

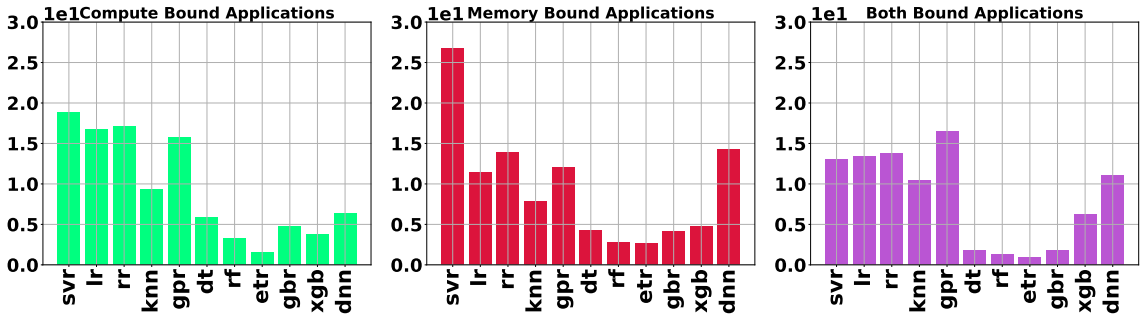


Figure 5.10: Mean MedAPE for all application types

5.4 Results from Paper : 3

In this section, we have performed several experiments to assess the performance of machine learning models for multivariate runtime and power prediction. We have also implemented transfer learning for cross-platform and cross-system multivariate prediction.

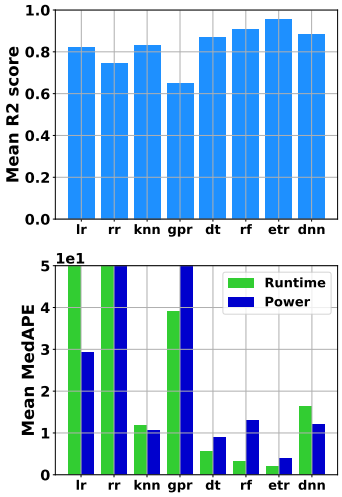


Figure 5.11: Mean MedAPE for all application types

5.4.1 Model Comparison for Multivariate Prediction

Fig. 5.11 shows the overall mean result for runtime and power prediction for all ML models listed in table 5.1 across all datasets as listed in table 3.1. Each model was trained with 80% training data and tested on 20% remaining data, and all metric scores were recorded for each dataset. Since the output is two dimensional, the R2 score is averaged for both runtime and power errors. Contrary, we calculated MedAPE separately each for runtime and power to observe and compare the difference in percentage errors.

1. **Two Uni-variate models each for runtime and power prediction** The first four models, namely lr, rr, knn, and gpr build a separate model for each output value like

Table 5.1: Machine Learning Models

Models Used	Abbreviations	True Multivariate
Linear Regressor	lr	No
Ridge Regr.[18]	rr	No
K-Nearest Neighbor Regressor[8]	knn	No
Gaussian Process Regressor.[32]	gpr	No
Decision Tree Regressor[23]	dt	Yes
Random Forest Regressor[9]	rf	Yes
Extra Trees Regressor[13]	etr	Yes
Deep Neural Network[36]	dnn	Yes

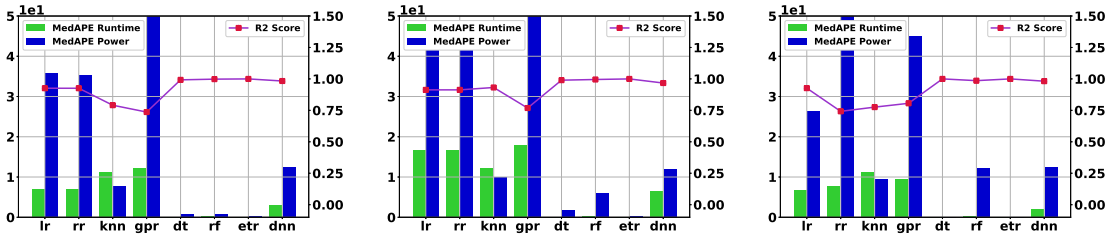
in our case, these models separately train two models internally (in Sklearn library) and give prediction results for both attributes runtime and power. The performance of these models can be observed in Fig. 5.11. It is observed that the knn model with mean MedAPE = 11.76% for runtime and mean MedAPE = 10.53% and mean R2 = 0.83 gives least prediction errors among the four models because knn supports non-linear solutions. Linear models lr and rr are giving more than 20% percentage errors. This is because of the non-linear relationship between the input hardware features and targets runtime and power. Gpr model assumes gaussian uncertainty for the output variable, and if that is wrong initially, the model doesn't predict the posterior distribution well.

2. **One model for multi-variate prediction** The last four models, namely dt, rf, etr, which are tree-based models and dnn, create one model while training and predicts both output variables simultaneously. The prediction results for these models are shown in Fig. 5.11. It is observed that the etr model with mean MedAPE = 2.14% for runtime and mean MedAPE = 3.94% for power and mean R2 = 0.95 gives best prediction accuracy among all other models including dnn. The tree-based models perform well because these models make decision rules with optimal splitting criterion at each node rather than considering correlation like other linear models. Etr performs better than dt and rf because the splits in the etr model are random, which leads to more diversified tree formation and works well in case of noisy features. We believe that Dnn is not able to outperform because it requires a larger number of training points to estimate the underlying function.

   One crucial observation is that multi-variate models (like tree-based models) perform better than uni-variate models for runtime and power prediction in diverse computer hardware systems.

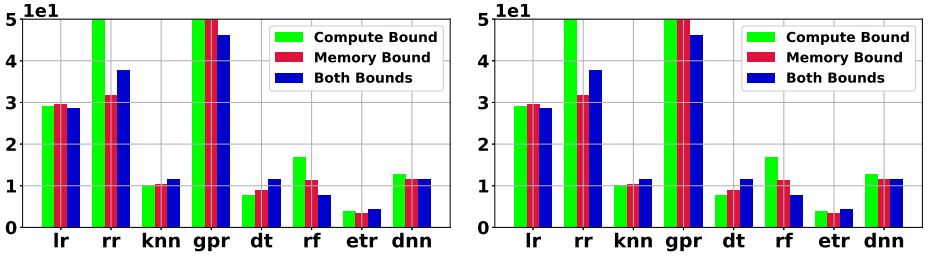
5.4.2 Performance Comparison concerning Application and System Types for multivariate prediction

We ran machine learning models for each dataset as listed in table 3.1 for performance and power prediction. A sample example illustrating the prediction accuracy for all models concerning each application type is shown in Fig. 5.12. We analyzed the results across different application types like compute, memory, and both bounds. Also, we examined model prediction results across datasets belonging to physical and simulated systems.



(a) SHA(CB) (b) Dijkstra(MB) (c) Stitch(CB+MB)

Figure 5.12: R2 score and MedAPE for each of one example Application types for simulated systems

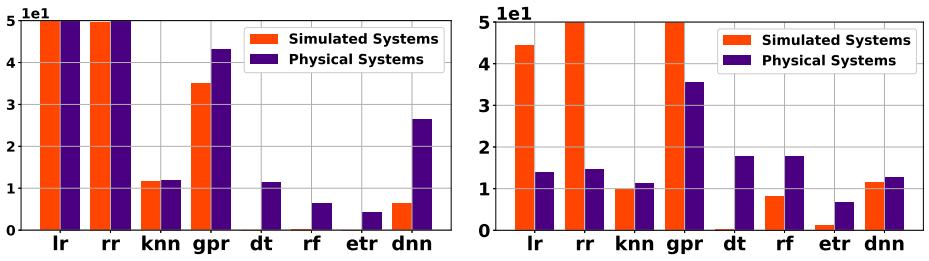


(a) Runtime Error (b) Power Error

Figure 5.13: Comparing model performance with respect to Application Type

1. **Application Type** In our dataset, diverse scientific applications are selected based on their computation and memory access patterns. Compute-bound applications like Montecarlocalcpi, perform a large number of computations per data element, whereas memory-bound applications like Matrix Multiplication require many data element per computation. Also, there are some applications like Quick Sort, which behave as both compute and memory-bound applications. We have taken four compute-bound applications, three memory-bound, and two both-bound applications. Fig. 5.13 shows the overall mean prediction results (Median Absolute Percentage Errors) across all datasets per application type for runtime in Fig. 5.13a and for power in Fig. 5.13b. The first observation is that the tree-based multi-variate

models dt, rf and etr with mean MedAPE = 3.77% for runtime and mean MedAPE = 8.47% for power are able to outperform all other models including deep neural network. We also observe that the error in power prediction is higher than runtime prediction error in all models. This is because power has a higher variance as compared to runtime. Finally, we observe that for non-linear models, the runtime prediction accuracy for compute-bound applications is higher than memory-bound or both-bound applications. This is because a larger number of data points from simulation systems use a simple Gem5[7] processor resulting in lower variation in runtime for compute-bound applications. On the other hand, the power prediction accuracy for compute-bound application is analogous to memory-bound applications because the power consumption of memory unit is insignificant compared to processor power resulting in similar power variations in all types of applications.



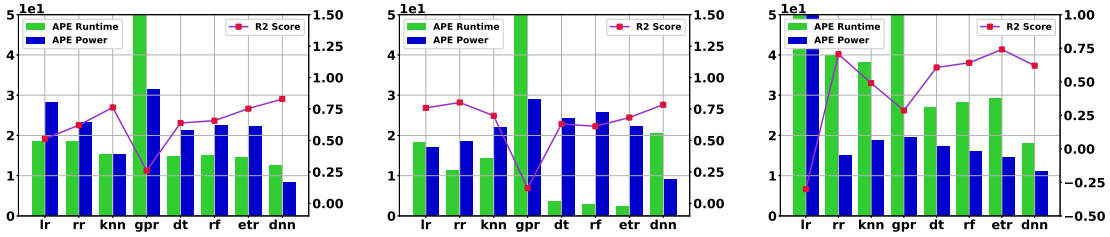
(a) Runtime Error (b) Power Error

Figure 5.14: Comparing model performance on Physical and Simulated Systems

2. **System Type** We have also given a comparison between prediction accuracy concerning the system type. Each scientific application was executed on both physical and simulated systems. We have used nine applications that were executed on both systems. The overall mean prediction errors for each model across all datasets are as shown in Fig. 5.14. The errors for runtime are shown in Fig. 5.14a and for power in Fig. 5.14b. It is observed that for simulated systems across tree-based models, the prediction accuracy with mean MedAPE = 0.096% for runtime and mean MedAPE = 3.21% for power is higher than physical systems with mean MedAPE = 7.31% for runtime and mean MedAPE = 14.10% for power. This is because physical systems are non-deterministic due to resource contention among multiple processes execution, whereas the simulation environment is deterministic, resulting in fewer variations in simulation systems than physical systems. Similar to the earlier section, here also we observe that the power prediction errors are more than runtime prediction errors due to higher variance in power.

5.4.3 Cross-Platform Multivariate prediction using Transfer Learning

For multivariate runtime and power prediction, we performed experimentation on Quick Sort simulation (2730 data points) dataset for section 1 and physical dataset (672 data points) for section 2 since it had a maximum number of data points. (Refer table 3.1). The Quick Sort simulation dataset is divided into 3 types of platforms, i.e., Intel Core (1440 data points), ARM (720 data points), and AMD (570 data points) platforms data. Whereas, the Quicksort Physical dataset is divided into Intel Core (360 data points) and Intel Xeon (312 data points) platforms data.



(a) Intel to ARM (b) Intel to AMD (c) Intel-Core to Intel-Xeon

Figure 5.15: Transfer Learning on Qsort Simulated Dataset (a and b) and Qsort Physical Dataset (c)

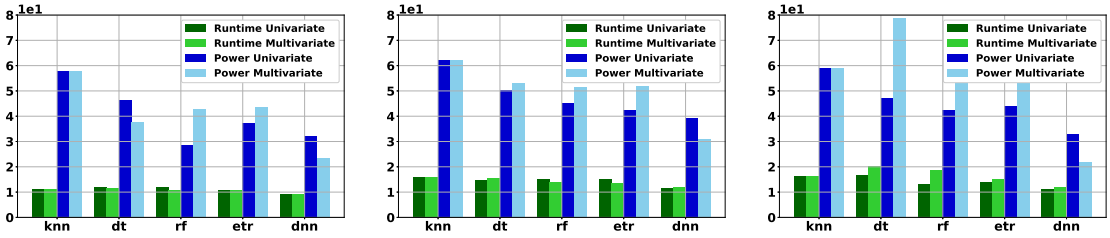
1. **Intel Core to ARM and AMD** In this section, we use transfer learning to predict ARM or AMD systems (platform-2) targets using Intel Core systems (platform-1) in a simulation environment. We took 100% data of platform-1 and 1% data of platform-2. We then trained the models (except dnn) only once on combined data (Since partial_fit() function is not available in Sklearn library for all models) and tested for the remaining 99% data. For deep neural network (dnn), we trained the model (epochs = 100 with batch_size = 10) as shown in Fig. 4.1b on 100% data of platform 1. Later, we retrained the model (epochs = 100 with batch_size = 1) by using 1% data of platform 2 after freezing all layers except the last 2 layers of the model and, finally, tested on remaining data of platform 2. The results for cross-platform prediction are shown in Fig. 5.15. From Fig. 5.15a, it is observed that dnn with R2 score = 0.83 including APE = 12.47% for runtime and APE = 8.32% for power outperforms all other models. Also, we achieved mean APE = 14.24% for runtime and mean APE = 18.61% for power prediction using transfer learning across dnn and tree-based models. Fig. 5.15b shows metrics for cross-platform Intel Core to AMD multivariate prediction for each model. It is observed that tree-based models with mean APE = 2.89% outperforms other models, including dnn in case of runtime whereas, dnn with mean APE = 8.98% outperforms other models in case of power prediction.

2. **Intel Core to Intel Xeon** In this section, we use transfer learning to predict Intel Xeon systems (platform-2) targets using Intel Core systems (platform-1) for physical systems. We took 100% data of platform-1 and 10% data of platform-2. We used 10% from platform-2 since the data points in the Quick Sort Physical dataset were still less than the simulation dataset to have an accurate prediction. We then trained the models on combined data and tested for the remaining 90% data of platform 2. For deep neural network (dnn), as explained in previous section, we trained the model (epochs = 200with batch_size = 1) as shown in Fig. 4.1b on 100% data of platform-1, later fine-tuned (epochs = 200 with batch_size = 1) by using 10% data of platform-2 and, finally tested on remaining data of platform-2. The results for cross platform prediction are shown in Fig. 5.15. It is observed from Fig. 5.15c that dnn with APE = 18.04% for runtime and APE = 11.08% for power outperforms all other models. The result is similar to the case of Intel-Core to ARM prediction.

5.4.4 Cross-System Multivariate and Univariate Prediction

As we have already experimented practicing transfer learning for cross-platform prediction, we also implemented it for cross-system prediction i.e., from simulated to physical systems. We performed experiments on three application dataset, namely SVM(CB) in Fig. 5.16a, tracking(MB) in Fig. 5.16b and, Stitch (CB+MB) in Fig. 5.16c to include each application type for observing model performance across varied application types. Moreover, we have compared the results of multivariate and univariate runtime and power prediction for cross-systems. For Univariate, we trained separate models each for runtime and power and then took the average of the results for comparison with multivariate results. For the implementation of transfer learning in machine learning models, we used 100% data of simulated systems and appended 10% data of physical systems to it. After that, we trained the models only once (Since partial_fit() function is not available in Sklearn library for all models) and then predicted the result for the remaining 90% data of physical systems. For the deep neural network, as shown in Fig. 4.1b, we used same method as described in section 5.4.3 with fine-tuning (Epochs = 100 with batch_size = 5 for both training and fine-tuning). The results for cross-system runtime and power prediction are shown in Fig. 5.16.

1. **Prediction results for cross-system for each application type** Fig. 5.16 shows prediction errors each for runtime and power across univariate and multivariate models. From Fig. 5.16a for compute-bound application, it is observed that dnn multivariate model outperforms other models with APE = 9.10% for runtime and with APE = 23.38% for power prediction. Moreover, from Fig. 5.16b for memory-bound application, it is observed that dnn multivariate model outperforms other models



(a) SVM (CB) (b) Tracking (MB) (c) Stitch (CB+MB)

Figure 5.16: APE error for Simulation to Physical Systems prediction using Transfer Learning

with APE = 11.81% for runtime and with APE = 31.03% for power prediction. Similarly, From Fig. 5.16c for compute and memory-bound application, it is observed that dnn multivariate model outperforms other models with APE = 12.11% for runtime and with APE = 22.05% for power prediction. From the results, it is clear that dnn multivariate models give the best prediction accuracy for both runtime and power. Furthermore, it is observed that there is almost a similar performance for runtime in both univariate and multivariate dnn models, whereas there is a significant increase in power prediction accuracy via dnn multivariate model.

2. **Multi-variate vs Uni-variate prediction** Fig. 5.16 shows the prediction errors for univariate and multivariate runtime and power prediction for each application type. It is observed that dnn is outperforming for cross-system prediction in both multivariate and univariate case. Also, it is seen that univariate models with mean APE = 13.99% (mean for all models except dnn across all three applications) for runtime and mean APE = 46.84% for power have better performance than multivariate models with mean APE = 14.52% for runtime and mean APE = 53.83% for power. Thus for cross-system prediction, univariate machine learning models except dnn perform better. Furthermore, in all three applications, dnn multivariate model with mean APE = 25.48% (across all datasets) outperforms dnn univariate model with mean APE = 34.81%

CHAPTER 6
Conclusion and Future Work

In this paper, we have experimented with different machine learning models for a wide variety of datasets consisting of three different application types and two different system types. We have done rapid experimentation to compare and conclude important observations to give an in-depth analysis of univariate and multivariate modeling. Furthermore, we have also used transfer learning approach for cross-domain predictions. The conclusions that can be made from the experiments performed are as follows:

6.1 Performance Analysis: Univariate

1. The tree-based models including tree-based bagging models dt, etr and rf outperform all other machine learning models including the deep learning variants.
2. After exhaustive comparison of physical and simulated system applications, it is concluded that models have higher accuracy on simulated systems.
3. 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.
4. Dimensionality reduction using PCA is an important pre-processing step and increases model accuracy.
5. Grid Search proved useful for optimal parameters selection and achieving better prediction accuracy.
6. Cross-systems performance prediction from simulation to physical systems using transfer learning yields less than 15% percentage errors using tree-based models.
7. 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.

8. 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.

9. Due to variability in processor manufacturing, prediction accuracy in memory-bound applications is higher than compute-bound applications.

6.2 Performance and Power Analysis: Multivariate

1. Multivariate Tree-based models namely dt, rf and etr outperform all other models including deep learning models. Also, multivariate models perform better than univariate models if the training and prediction are made on the same application dataset.

2. Among univariate models, knn gives the best performance with less than 12% percentage errors. Also, among multivariate models, etr gives the best performance with less than 4% percentage errors.

3. The errors in power prediction are more than runtime prediction due to higher variations in power.

4. Simulated systems give less multivariate prediction errors than physical systems due to the deterministic nature of simulated systems.

5. For cross-platform prediction, dnn multivariate model and tree-based models outperform all other models for power and runtime, respectively.

6. For cross-system prediction, dnn multivariate outperforms all other uni or multivariate models.

Our future work will include more experimentation for building efficient multivariate models to include the cost in addition to performance and power and contribute more data sets to the research community. We also plan to experiment with the data augmentation technique to add more data to physical systems dataset.

References

[1] B. C. R. K. A. A. Amit Mankodi, Amit Bhatt. Evaluating machine learning models for disparate computer systemsperformance prediction. 2020.

[2] B. C. R. K. A. A. Amit Mankodi, Amit Bhatt. Modeling performance and power on disparate platforms using transfer learning with machine learning models. 2020.

[3] N. Ardalani, C. Lestourgeon, K. Sankaralingam, and X. Zhu. Cross-architecture performance prediction (XAPP) using CPU code to predict GPU performance. In *Proceedings of the Annual International Symposium on Microarchitecture, MICRO*, volume 05-09-Dece, pages 725–737. IEEE Computer Society, dec 2015.

[4] M. Awad and R. Khanna. *Support Vector Regression*, pages 67–80. Apress, Berkeley, CA, 2015.

[5] 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.*, 5(3):63–73, Sept. 1991.

[6] I. Baldini, S. J. Fink, and E. Altman. Predicting GPU Performance from CPU Runs Using Machine Learning. In *2014 IEEE 26th International Symposium on Computer Architecture and High Performance Computing*, pages 254–261, oct 2014.

[7] N. Binkert, B. Beckmann, G. Black, S. K. Reinhardt, A. Saidi, A. Basu, J. Hestness, D. R. Hower, T. Krishna, S. Sardashti, R. Sen, K. Sewell, M. Shoaib, N. Vaish, M. D. Hill, and D. A. Wood. The gem5 simulator. *SIGARCH Comput. Archit. News*, 39(2):1–7, Aug. 2011.

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

[9] L. Breiman. Random Forests. *Machine Learning*, 45(1):5–32, 2001.

[10] 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*, pages 785–794, 2016. arXiv: 1603.02754.

[11] 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.

[12] J. H. Friedman. Stochastic gradient boosting. *Computational Statistics & Data Analysis*, 38(4):367–378, Feb. 2002.

[13] P. Geurts, D. Ernst, and L. Wehenkel. Extremely randomized trees. *Machine Learning*, 63(1):3–42, Apr. 2006.

[14] 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)*, pages 1–6, Nov 2018.

[15] 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*, 47:112 – 120, 2016.

[16] M. R. Guthaus, J. S. Ringenberg, D. Ernst, T. M. Austin, T. Mudge, and R. B. Brown. Mibench: A free, commercially representative embedded benchmark suite. In *Proceedings of the Fourth Annual IEEE International Workshop on Workload Characterization. WWC-4 (Cat. No.01EX538)*, pages 3–14, 2001.

[17] 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. Technical Report SAND2009-5574, Sandia National Laboratories, 2009.

[18] A. E. Hoerl and R. W. Kennard. Ridge regression: Biased estimation for nonorthogonal problems. *Technometrics*, 12(1):55–67, 1970.

[19] P. Jamshidi, M. Velez, C. Kästner, N. Siegmund, and P. Kawthekar. Transfer learning for improving model predictions in highly configurable software. In *Proceedings of the 12th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*, SEAMS '17, page 31–41. IEEE Press, 2017.

[20] S. Kawaguchi and T. Yachi. Adaptive power efficiency control by computer power consumption prediction using performance counters. *IEEE Transactions on Industry Applications*, 52(1):407–413, jan 2016.

[21] A. Krizhevsky, I. Sutskever, and G. Hinton. Imagenet classification with deep convolutional neural networks. *Neural Information Processing Systems*, 25, 01 2012.

[22] Z. Lai, K. T. Lam, C. L. Wang, and J. Su. PoweRock: Power Modeling and Flexible Dynamic Power Management for Many-Core Architectures. *IEEE Systems Journal*, 11(2):600–612, jun 2017.

[23] W.-Y. Loh. Classification and regression trees. *WIREs Data Mining and Knowledge Discovery*, 1(1):14–23, 2011.

[24] 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)*, pages 824–827, Dec 2018.

[25] J. Ma, G. Yan, Y. Han, and X. Li. An analytical framework for estimating scale-out and scale-up power efficiency of heterogeneous manycores. *IEEE Transactions on Computers*, 65(2):367–381, feb 2016.

[26] 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)*, pages 33–44, Nov 2018.

[27] 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*, pages 33–44. Institute of Electrical and Electronics Engineers Inc., feb 2019.

[28] G. Mariani, A. Anghel, R. Jongerius, and G. Dittmann. Predicting cloud performance for hpc applications before deployment. *Future Generation Computer Systems*, 87:618 – 628, 2018.

[29] F. Moradi, R. Stadler, and A. Johnsson. Performance prediction in dynamic clouds using transfer learning. In *2019 IFIP/IEEE Symposium on Integrated Network and Service Management (IM)*, pages 242–250, 2019.

[30] 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*, 12:2825–2830, 2011.

[31] A. B. B. C. A. A. Rajat Kumar, Amit Mankodi. Cross-platform performance prediction with transfer learning using machine learning. 2020.

[32] B. O. Rasmussen, Carl Edward, U. von Luxburg, and G. Rätsch, editors. *Gaussian Processes in Machine Learning*, pages 63–71. Springer Berlin Heidelberg, Berlin, Heidelberg, 2004.

[33] T. J. Santner, B. J. Williams, and W. I. Notz. The design and analysis of computer experiments. *Springer Series in Statistics*, 2003.

[34] A. Sirbu and O. Babaoglu. Predicting System-level Power for a Hybrid Supercomputer. In *Proceedings of the 2016 International Conference on High Performance Computing & Simulation (HPCS 2016) : July 18-22, 2016, Innsbruck, Austria*, 2016.

[35] 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)*, pages 55–64, Oct 2009.

[36] H. Wang and B. Raj. On the origin of deep learning. 02 2017.

[37] X. Zheng, L. K. John, and A. Gerstlauer. LACross: Learning-Based Analytical Cross-Platform Performance and Power Prediction. *International Journal of Parallel Programming*, 45(6):1488–1514, dec 2017.