Modeling Performance and Power Matrix of Disparate Computer Systems using Machine Learning Techniques
(Modeling Computer Systems Selection)

by

AMIT MANKODI
201621001

A Thesis Submitted in Partial Fulfilment of the Requirements for the Degree of

DOCTOR OF PHILOSOPHY
to

DHIRUBHAI AMBANI INSTITUTE OF INFORMATION AND COMMUNICATION TECHNOLOGY

March, 2022

Declaration

I hereby declare that

i) the thesis comprises of my original work towards the degree of Doctor of Philosophy 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.

Amit Mankodi

Certificate

This is to certify that the thesis work entitled MODELING PERFORMANCE AND POWER MATRIX OF DISPARATE COMPUTER SYSTEMS USING MACHINE LEARNING TECHNIQUES has been carried out by AMIT MANKODI for the degree of Doctor of Philosophy at *Dhirubhai Ambani Institute of Information and Communication Technology* under our supervision.

Prof Amit Bhatt
Thesis Supervisor

Prof Bhaskar Chaudhury
Thesis Co-Supervisor

Acknowledgments

I would like to express my deepest gratitude to my mentors Professor Amit Bhatt and Professor Bhaskar Chaudhury for their continuous support. Without their guidance over these years, this work would not have been possible. I sincerely appreciate their input in the selection of my research area, the selection of conferences and journals, and for providing me with their valuable feedback and correcting me on many occasions and encouraging me to improve the thesis work both technically and syntactically.

I would also like to thank my committee members Professor Arnab Ray, Professor Rajib Lochan Das and Professor Biswajit Mishra for their valuable input during the research seminars. Being unfamiliar with machine learning, it was challenging to understand and apply machine learning to performance and power prediction models. I am thankful to many Professors from DA-IICT who guided me to get a better grip on the area of machine learning. A special thank you to Professor Binita Desai for spending many hours reviewing and providing valuable input for improving important sections of the thesis.

I would also like to thank the lab staff for providing me access to the required computer systems and other hardware resources. Thanks to library staff especially Mr. Manish Mankad for helping me with access to articles and productive searches during the literature review.

I owe my wife Alpa, a big thank you. Without her unconditional encouragement and patience, I would not have succeeded and overcome disappointments during my Ph.D. years. Finally, I would like to thank my parents, Kishor and Nayana, for instilling in me the value of higher education and giving me their unequivocal support throughout my life.

Contents

Abstract **xiv**

List of Tables **xv**

List of Figures **xvii**

1 Introduction **1**

1.1 Performance and Power Prediction Modeling 3

1.1.1 Cross Performance and Power Prediction 4

1.1.2 Prediction with Transfer Learning 5

1.2 Modeling Challenges Due To an Applications Dependence on System Features 6

1.3 Motivation: Leverage Simulated Systems To Model Performance and Power for Physical Systems 8

1.4 Thesis Problem Statement and Proposed Solutions 10

1.5 Thesis Contribution 12

1.6 Thesis Organization 18

2 Learning Based Performance Prediction of Applications on Disparate Computer Systems **21**

2.1 Overview 21

2.2 Learning Based Performance Prediction Model 25

2.3 Dataset Construction 28

2.3.1 Construction of Simulation-based Systems in Gem5 Simulator 28

2.3.1.1 Selection of Systems 29

2.3.1.2 Challenges During Construction of Simulated Systems in Gem5 . . . 30
2.3.1.2.1 Challenge 1: Gem5 Memory Support . . . 30
2.3.1.2.2 Challenge 2: Gem5 Cache Support . . . 32
2.3.1.2.3 Challenge 3: Gem5 Processor Support . . . 36
2.3.2 Selection of Physical Systems . . . 37
2.3.3 Applications for Workloads . . . 37
2.4 Feature Encoding and Normalization . . . 39
2.4.1 Encoding Categorical Features . . . 39
2.4.2 Normalizing Feature Set . . . 40
2.5 Training and Prediction Model . . . 41
2.5.1 Linear Regression Model . . . 41
2.5.2 Decision Tree Regression Model . . . 41
2.6 Experimental Results and Analysis . . . 42
2.6.1 Prediction Accuracy for Simulated Computer Systems . . . 43
2.6.2 Prediction Accuracy for Physical Computer Systems . . . 45
2.6.3 Model Evaluation using Physical Systems Results . . . 46
2.6.3.1 Linear Regression vs Decision Tree . . . 46
2.6.3.2 Train vs Test Dataset Split Ratio . . . 50
2.6.3.3 Additional Architecture Features for Prediction Improvement . . . 52
2.6.4 Performance Prediction of Mantevo mini-applications and NAS Parallel Benchmarks . . . 52
2.7 Threats to Validity and Limitations . . . 55
2.8 Summary . . . 56

3 Evaluating Machine Learning Models for Disparate Computer Systems Performance Prediction 60

3.1 Overview . . . 60
3.2 Dataset Preparation . . . 62
3.2.1 Applications Selection for Workload . . . 62
3.2.2 Computer Systems Selection . . . 62

3.3 Machine Learning Models . . . 64
3.3.1 Support Vector Regressor (svr) . . . 64
3.3.2 Multiple Linear Regression (lr) . . . 64
3.3.3 Ridge Regression (rr) . . . 65
3.3.4 K Nearest Neighbors (knn) . . . 65
3.3.5 Gaussian Process Regressor (gpr) . . . 65
3.3.6 Decision Trees Regressor (dt) . . . 65
3.3.7 Random Forest Regressor (rf) . . . 66
3.3.8 Extremely Randomized Trees (etr) . . . 66
3.3.9 Gradient Boosting Regressor (gbr) . . . 66
3.3.10 XGBoost (xgb) . . . 67
3.3.11 Deep Neural Networks . . . 67
3.4 Experimental Evaluation/Results . . . 68
3.4.1 Model comparison for performance prediction . . . 69
3.4.2 Estimating the Effect of Performance concerning Compute-Bound and Memory-Bound Applications . . . 71
3.4.3 Comparing model performance on Physical and Simulated Systems . . . 72
3.5 Evaluation of Neural Network Models . . . 75
3.5.1 Neural Network Models . . . 75
3.5.2 Experimental Evaluation of Neural Network Models . . . 78
3.5.2.1 Impact of Data Scaling . . . 79
3.5.2.2 One-Layer vs Multi-Layer Neural Network . . . 80
3.5.2.2.1 Number of Neurons . . . 82
3.5.2.2.2 Optimizer Selection . . . 82
3.6 Summary . . . 83

4 Multivariate Performance and Power Prediction of Applications on Simulated Systems 86
4.1 Overview . . . 86
4.2 Multivariate Prediction Model . . . 89
4.3 Experimental Setup . . . 91

4.3.1 Dataset Construction . . . 91
4.3.2 Collecting Dynamic Runtime Power . . . 92
4.3.3 Training and Prediction Model . . . 93
4.4 Results . . . 95
4.4.1 Cross-Validation Error . . . 96
4.4.2 Prediction Accuracy . . . 98
4.5 Summary . . . 100

5 Cross Prediction with Scaling using Machine Learning 102
5.1 Performance Prediction of Physical Computer Systems Using Simulated Systems . . . 102
5.1.1 Overview . . . 102
5.1.2 Cross Performance Prediction Model . . . 104
5.1.3 Experimental Setup . . . 105
5.1.3.1 Simulation-based Models for Training . . . 106
5.1.3.2 Applications Selection for Workload . . . 106
5.1.3.3 Physical Computer Systems for Cross Performance Prediction . . . 107
5.1.4 Results . . . 108
5.1.4.1 Simulation-based Performance Prediction Accuracy 109
5.1.4.2 Cross Performance Prediction Accuracy . . . 110
5.1.5 Summary . . . 112
5.2 Predicting Physical Computer Systems Performance and Power from Simulated Systems using Machine Learning Model . . . 113
5.2.1 Overview . . . 113
5.2.2 Methodology and Model . . . 116
5.2.2.1 Methodology . . . 116
5.2.2.2 Cross Performance and Power Prediction with Scaling Model . . . 117
5.2.2.3 Determining Scaling Factor . . . 119
5.2.3 Experimental Setup and Dataset Construction . . . 120
5.2.3.1 Application Selection for Workload . . . 121

5.2.3.2 Simulated Systems for Training 122
5.2.3.3 Physical Computer Systems for Cross Performance and Power Prediction 122
5.2.4 Results 123
5.2.4.1 Simulation-based Performance Prediction Accuracy 124
5.2.4.2 Cross Performance and Power Prediction with Scaling Model Accuracy 124
5.2.4.2.1 Cross Predicted Targets (Yppred) vs Actual Targets from Gem5-based Physical Systems (Ysp) 125
5.2.4.2.2 Cross Predicted Targets (Yppred) vs Actual Targets from Physical Systems (Yp) . 125
5.2.5 Selection of Computer Systems 131
5.2.6 Summary 132

6 Cross Prediction with Transfer Learning using Machine Learning 133

6.1 Cross-Platform Performance Prediction with Transfer Learning using Machine Learning 133
6.1.1 Overview 133
6.1.2 Related Work 134
6.1.3 Dataset Preparation 135
6.1.3.1 Applications Selection for Workload 135
6.1.3.2 Computer Systems Selection 135
6.1.4 Models and Techniques Used 137
6.1.4.1 Machine Learning Models 137
6.1.4.2 Grid Search 138
6.1.4.3 Transfer Learning 139
6.1.5 Experimental Evaluation 139
6.1.5.1 Effect of Dimensionality Reduction using PCA . . 140
6.1.5.2 Effect of Grid Search on Model Performance 141
6.1.5.3 Cross Prediction using Transfer Learning 141

6.1.5.3.1 Cross-Systems Prediction: Simulated Systems to Physical Systems 141
6.1.5.3.2 Cross-Platforms Prediction: x86-based Systems to ARM-based Systems 143
6.1.5.3.3 Cross-Platform Prediction: Intel Core to Intel Xeon 144
6.1.5.4 Model Comparison for Performance Prediction . . 144
6.1.5.5 Estimating the Effect of Performance with respect to Compute-Bound, Memory-Bound and Compute-plus-Memory-Bound Applications 147
6.1.6 Summary 149
6.2 Modeling Performance and Power on Disparate Platforms using Transfer Learning with Machine Learning Models 150
6.2.1 Overview 150
6.2.2 Dataset Preparation 152
6.2.2.1 Applications Selection for Workload 152
6.2.2.2 Computer Systems Selection 152
6.2.3 Machine Learning Models and Techniques Used 154
6.2.3.1 Models Description 154
6.2.3.2 Transfer Learning 155
6.2.4 Experimental Evaluation 156
6.2.4.1 Model Comparison for Multivariate Prediction . . 157
6.2.4.1.1 Prediction Accuracy of Univariate Models: 157
6.2.4.1.2 Prediction Accuracy of Multivariate Models: 158
6.2.4.2 Multivariate Prediction Accuracy per Application Types and System Types 159
6.2.4.2.1 Prediction Accuracy per Application Type: 159
6.2.4.2.2 Prediction Accuracy per System Type: . . 160

6.2.4.3 Cross-Platform Multivariate Prediction using Transfer Learning . . . 161
6.2.4.3.1 Cross-Platform Prediction: Intel Core to ARM and AMD . . . 161
6.2.4.3.2 Cross-Platform Prediction: Intel Core to Intel Xeon . . . 162
6.2.4.4 Cross-Systems Multivariate and Univariate Prediction using Transfer Learning . . . 163
6.2.4.4.1 Prediction Accuracy per Application Type: . . . 164
6.2.4.4.2 Prediction Accuracy of Univariate and Multivariate Implementation: . . . 164
6.2.5 Summary . . . 165

7 Cross Prediction Models: Scaling vs. Transfer Learning 166
7.1 Overview . . . 166
7.2 Related Work . . . 170
7.2.1 Cross-Platform Performance Prediction . . . 170
7.2.2 Performance Prediction with Transfer Learning . . . 172
7.3 Cross Performance Prediction Models . . . 174
7.3.1 Cross Performance Prediction with Scaling . . . 174
7.3.2 Cross Performance Prediction with Transfer Learning . . . 176
7.4 Experiment Platforms . . . 177
7.4.1 Simulated Systems . . . 177
7.4.2 Applications Selection for Workload . . . 177
7.4.3 Physical Systems for Cross Performance Prediction . . . 178
7.4.4 Decision Tree Regression Machine Learning Model . . . 179
7.5 Cross Performance Prediction with Scaling . . . 181
7.5.1 Experimental Details . . . 181
7.5.2 Results . . . 182
7.5.2.1 Prediction Accuracy of Decision Tree Machine Learning Model . . . 183

7.5.2.2 Cross Performance Prediction Accuracy with and without Scaling . . . 185
7.6 Cross Performance Prediction with Transfer Learning . . . 186
7.6.1 Experimental Details . . . 187
7.6.2 Results . . . 189
7.7 Prediction Accuracy: Scaling vs Transfer Learning . . . 191
7.8 Summary . . . 192

8 Summary, Conclusions and Future Work 194
8.1 Summary and Conclusions . . . 194
8.2 Future Work Direction . . . 197
8.2.1 Improve Model Accuracy . . . 197
8.2.2 Adapting to Complex Application . . . 198
8.2.3 Explore Systems Architectures . . . 198

References 200

Appendix A Glossary 216

Abstract

In the last couple of decades, there has been an exponential growth in the processor, cache, and memory features of computer systems. These hardware features play a vital role in determining the performance and power of a software application when executed on different computer systems. Furthermore, any minor alterations in hardware features or applications can alter and impact the performance and power consumption. Compute-intensive (compute-bound) applications have a higher dependence on processor features, while data-intensive (memory-bound) applications have a higher dependence on memory features. To match the customized budgets in performance and power, selecting computer systems with appropriate hardware features (processor, cache, and memory) becomes extremely essential. To adhere to user-specific budgets, selecting computer systems requires access to physical systems to gather performance and power utilization data. To expect a user to have access to physical systems to achieve this task is prohibitive in cost; therefore, it becomes essential to develop a virtual model which would obviate the need for physical systems.

Researchers have used system-level simulators for decades to build simulated computer systems using processor, cache, and memory features to provide estimates of performance and power. In one approach, building virtual systems using a full-system simulator (FSS), provides the closest possible estimate of performance and power measurement to a physical system. In the recent past, machine learning algorithms have been trained on the above-mentioned accurate FSS models to predict performance and power for varying features in similar systems, achieving fairly accurate results. However, building multiple computer systems in a full-system simulator is complex and an extremely slow process. The prob-

lem gets compounded due to the fact that access to such accurate simulators is limited.

However, there is an alternative approach of utilizing the open-source gem5 simulator using its emulation mode to rapidly build simulated systems. Unfortunately, it compromises the measurement accuracy in performance and power as compared to FSS models. When these results are used to train any machine learning algorithm, the predictions would be slightly inaccurate compared to those trained using FSS models. To make this approach useful, one needs to reduce the inaccuracy of the predictions that are introduced due to the nature and design of the gem5 functionality and as a consequence of this, the variation introduced due to the types of applications, whether it is compute-intensive or data-intensive.

This dissertation undertakes the above-mentioned challenge of whether one can effectively combine the speed of the open-access gem5 simulated system along with the accuracy of a physical system to acquire accurate machine learning predictions. If this challenge is met, a user would be able to successfully select a system either in the cloud or in the real world to run applications within ones' power and performance budget.

In our proposed methodology, we first created several gem5 models using the emulation mode for available systems with varying features like the type of processors (Instruction Set Architecture, speed and cache configuration), type of memory its speed and size. We executed compute-intensive and data-intensive benchmark applications to these models to procure performance results. In the second step, 80% of the models, generated using the gem5 simulator in the emulation mode, were used to train machine learning algorithms like linear, support vector, Gaussian, tree-based and neural network. The remaining 20% models were used for the purpose of performance prediction. It was found that the tree-based algorithm predicted the closest performance values compared to the simulated systems' results obtained using the above-mentioned gem5 model. We subsequently used hardware configuration and application execution statistics data generated by the gem5 model and fed it to the Multicore Power Area and Timing (McPAT) modeling tool which would estimate power usage.

To check the accuracy of the gem5 simulator results, the above-mentioned benchmark applications were fed to real systems with identical features. The given application code was modified to invoke the Performance Application Programming Interface (PAPI) function to measure the power consumption. There was a sizeable difference between the results of the gem5 model and the real system in terms of performance and power.

We conceptualized the idea of using scaling and transfer learning in the context of bridging the difference between predicted values to actual values. We proposed a scaling technique that can establish an application-specific scaling factor using a correlation coefficient between hardware features and performance/power. This scaling factor would capture the difference and apply it to a set of predicted values to conform to those of the physical system. The results demonstrate that for selected benchmark applications the scaling technique achieves a prediction accuracy of 75%-90% for performance and 60%-95% for power. The accuracy of the results validates that the scaling technique effectively attempts to bring predicted performance and power values closer to that of physical systems to enable the selection of an appropriate computer system(s).

Another method to achieve better prediction values is to develop a model based on the existing transfer learning technique. To use the transfer learning method, we train the decision tree algorithm based on two sets of data; one, from a simulated system and the second from a closely matching physical system. Using trained models, we attempt to predict the performance and power of the target physical system. The target system is different from the source physical system used for training the machine learning algorithm. This model uses performance and power from a source physical system during training to bring predicted values closer to that of the target system. The results from the transfer learning technique for selected benchmark applications display the mean prediction accuracy for different target systems to be between 10% to 50%.

In this work, we have demonstrated that our proposed techniques, scaling and transfer learning, are effective in estimating fairly accurate performance and power values for the physical system using the predicted values from a machine

learning model trained on a gem5 simulated systems dataset. Therefore, these techniques provide a method to estimate performance and power values for physical computer systems, with known hardware features, without a need for access to these systems. With estimated performance and power values coupled with hardware features of the physical systems, we can select system(s) based on user-provided budget/s of performance and power.

List of Tables

2.1 Computer Systems Built in Gem5 Simulator 29
2.2 Supported Memory Modules in Gem5 Simulator 30
2.3 Memory Modules Added in Gem5 Simulator 31
2.4 Gem5 Simulator O3CPU Model Features 37
2.5 Physical Computer Systems used for Matrix Multiplication, Monte Carlo and Quicksort 38
2.6 Physical Computer Systems used for EP, MG and miniFE 54

3.1 Physical Computer Systems used for Evaluation of Machine Learning Models 63
3.2 Applications used as workloads for Evaluation of Machine Learning Models 63
3.3 Model Summary of DNN Variants 67

4.1 Applications used as workloads for Multivariate Model 92
4.2 Prediction Accuracy for Different Train-Test Ratio 95

5.1 Applications used as workloads for Cross Performance Prediction Model with Scaling 106
5.2 Physical Computer Systems for Cross Performance Prediction Model with Scaling 108
5.3 Applications used as workloads for Cross Performance and Power Prediction Model with Scaling 121
5.4 Runtime Minor Factor Based on Pearson Correlation Coefficient . . 122
5.5 Physical Computer Systems for Cross Performance and Prediction Model with Scaling 123

6.1 Application and System Types used for Cross Performance Prediction Model with Transfer Learning 136
6.2 Physical Computer Systems for Cross Performance Prediction Model with Transfer Learning 137
6.3 Machine Learning Models for Cross Performance Prediction Model with Transfer Learning 138
6.4 Application and System Types used for Cross Performance and Power Prediction Model with Transfer Learning 153
6.5 Physical Computer Systems for Cross Performance and Power Prediction Model with Transfer Learning 154
6.6 Machine Learning Models for Cross Performance and Power Prediction Model with Transfer Learning 155

7.1 Applications used as workloads used for Comparison of Scaling versus Transfer Learning Cross Performance Models 178
7.2 Physical Computer Systems used for Comparison of Scaling versus Transfer Learning Cross Performance Models 179
7.3 True/Actual and Predicted Performance for Simulated Systems with Percentage Error 184

A.1 Glossary 216

List of Figures

1.1 Problem Formulation 2

1.2 Cache-aware Roofline Model [Figure taken from PICSAR application article [1] which uses cache aware roofline model from [2]] . . 7

2.1 Data Driven Learning-Based Performance Prediction Model 27

2.2 Gem5 Simulator System Cache Configuration 33

2.3 Linear Regression (LR) Prediction Error for Simulated Systems Test Architectures per Problem Size (a) Matrix Multiplication [Pearson Coeff: 90.28% to 92.83%], (b) Quicksort[Pearson Coeff: 93.63% to 94.17%], (c) Monte Carlo PI Calculation [Pearson Coeff: 94.87%], . 44

2.4 Decision Tree Regression (DTR) Prediction Error for Simulated Systems Test Architectures per Problem Size (a) Matrix Multiplication [Pearson Coeff: 100%], (b) Quicksort [Pearson Coeff: 100%], (c) Monte Carlo PI Calculation [Pearson Coeff: 100%] 45

2.5 Linear Regression (LR) Prediction Error for Physical Systems Test Architectures per Problem Size (a) Matrix Multiplication [Pearson Coeff: 94.84% to 96.17%], (b) Quicksort [Pearson Coeff: 91.67% to 99.88%], (c) Monte Carlo PI Calculation [Pearson Coeff: 98.82% to 98.85%] 47

2.6 Decision Tree Regression (DTR) Prediction Error for Physical Systems Test Architectures per Problem Size (a) Matrix Multiplication [Pearson Coeff: 97.83% to 99.40%], (b) Quicksort [Pearson Coeff: 91.82% to 99.98%], (c) Monte Carlo PI Calculation [Pearson Coeff: 98.52% to 98.87%] 48

2.7 Physical Systems: LR and DTR Median Error per Problem Size . . . 49
2.8 Input Features vs Runtime 49
2.9 Median absolute percentage error for different train-test split (a)-(c) Matrix Multiplication (d)-(f)Quicksort (g)-(i) Monte Carlo 51
2.10 Comparison of Median and Mean absolute percentage error for Monte Carlo with and w/o Bus Speed 53
2.11 miniFE, EP and MG Prediction Errors Per Problem Size 55

3.1 Mean of R^2, MedAPE and MSE values per model for all datsets . . 69
3.2 R^2 score and MedAPE for Compute-Bound Applications 72
3.3 R^2 score and MedAPE for Memory-Bound Applications 73
3.4 Mean MedAPE for both CB and MB Applications 73
3.5 Comparing model performance on Physical and Simulated Systems 74
3.6 Hardware Features vs Runtime 76
3.7 Fully-Connected Neural Network Models 77
3.8 Scaling Effect on Performance Prediction on Neural Network Model 80
3.9 Cross-Validation Prediction Error for One-Layer (OL) VS Multi-Layer (ML) Neural Network 81
3.10 Variations in Runtime with Respect to Mean 82
3.11 Effect of Neurons on Performance Prediction 83
3.12 Effect of Optimizers on Performance Prediction 84

4.1 Learning-Based Performance and Power Prediction Model 90
4.2 Power Generation & Collection Process for Simulated Systems . . . 93
4.3 Cross-Validation Prediction Error Per Problem Size 96
4.4 Samples Distribution over Runtime vs Power 97
4.5 Samples Distribution over Runtime vs Power for Benchmarks . . . 98
4.6 Prediction Accuracy for Test Simulated Systems 99
4.7 Prediction Accuracy of Test Simulated Systems for Benchmarks . . 100

5.1 Cross Performance Prediction Framework 105
5.2 Pearson Correlation Coefficient of Hardware Features with Runtime 107
5.3 Cross-Validation Mean Percentage Error 109

5.4 Simulator Runtime vs Cross Performance Prediction Runtime . . . 111
5.5 Physical System Runtime vs Cross Performance Prediction Runtime 112
5.6 Cross Performance and Power Prediction with Scaling Model . . . 118
5.7 Cross-Validation Mean Percentage Error (R)-Runtime, (P)-Power . 124
5.8 Simulator Runtime (μs) vs Cross Predicted Runtime (μs) 126
5.9 Simulator Power (W) vs Cross Predicted Power (W) 127
5.10 Variations in Runtime and Power With Respect To Mean 127
5.11 mser (compute-bound) vs tracking (memory-bound) power histogram 127
5.12 Physical System Runtime (μs) vs Cross Predicted Runtime (μs) . . . 128
5.13 Physical System Power (W) vs Cross Predicted Power (W) 129
5.14 System Runtime vs Power 132

6.1 DNN Model Summary 139
6.2 Application of PCA and Grid Search Techniques 142
6.3 Results for Training on Simulated System's Data and Testing on Physical System's Data 143
6.4 Results for Training and Testing for Cross-Platforms Prediction . . 145
6.5 Mean Prediction Scores per ML model across all datasets 145
6.6 R^2 score and MedAPE for Compute-Bound Applications 147
6.7 R^2 score and MedAPE for Memory-Bound Applications 148
6.8 R^2 score and MedAPE for Compute-Plus-Memory-Bound Applications 148
6.9 Mean MedAPE for all application types 149
6.10 DNN Model Summary 156
6.11 Mean of R^2 and MedAPE values per model across all datasets . . . 158
6.12 R^2 score and MedAPE for each of the example Application Types for Simulated Systems 159
6.13 Comparing Model Performance with respect to Application Type . 160
6.14 Comparing Model Performance on Physical and Simulated Systems 161
6.15 Transfer Learning on Quicksort Simulated Dataset (a and b) and Quicksort Physical Dataset (c) 163

6.16 APE error for Simulated Systems to Physical Systems Prediction using Transfer Learning 164

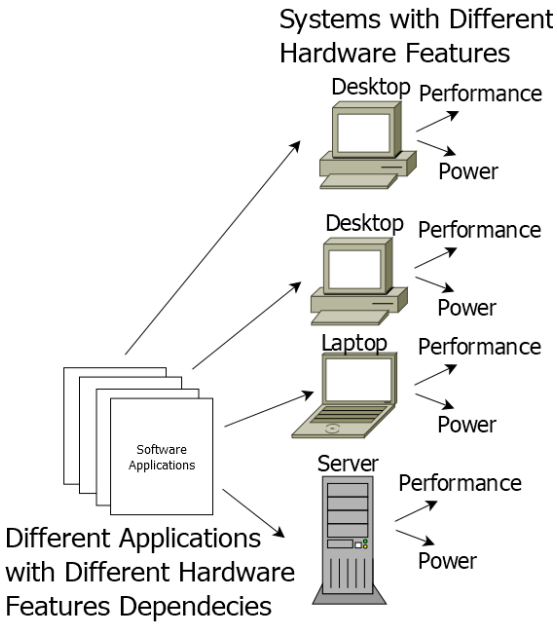
7.1 Cross Performance Prediction Model with Scaling 175
7.2 Cross Performance Prediction Model with Transfer Learning 175
7.3 Simulator Runtime vs Cross Performance Prediction Runtime . . . 183
7.4 Decision Tree Rules Determining Predicted Runtime 185
7.5 Physical System Runtime vs Cross Performance Prediction Runtime in Scaling Model 186
7.6 Prediction from each target system from different source systems in Transfer Learning Model 190
7.7 Comparison of Prediction Error for Transfer Learning vs Scaling . . 191

CHAPTER 1

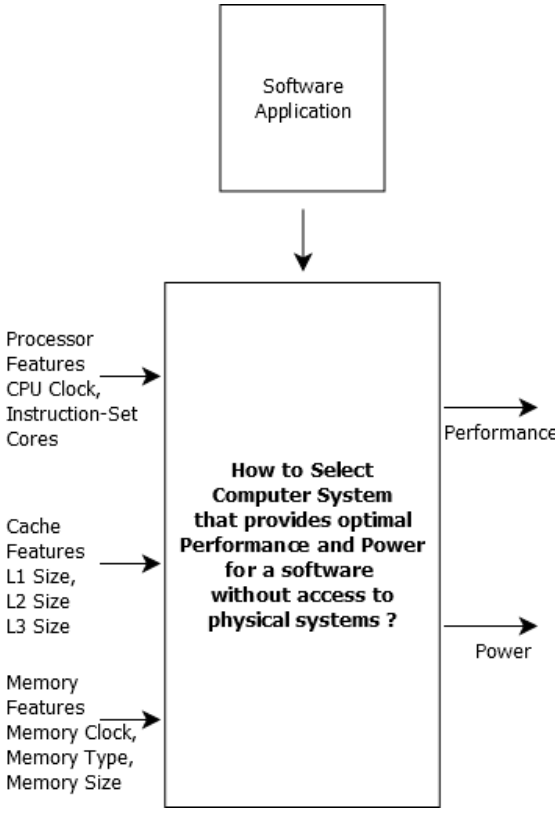
Introduction

Technological advancements in hardware features of computer systems, namely processor, cache, and memory, provide us with many options to acquire computer system(s) and they can be sourced either from commercially-off-the-self (COTS) or the cloud. These hardware features of computer systems largely determine the performance (execution runtime) and power consumption of a software application when executed on a computer system, as shown in figure 1.1a. Thus, a software application executed on computer systems with varied hardware features results in dissimilar performance and power consumption. Furthermore, various types of applications depend on different hardware features, resulting in an increased dissimilarity in performance and power consumption. For example, processor features have a greater influence on the performance and power of compute-intensive (compute-bound) applications, while memory features have a larger impact on the data-intensive (memory-bound) application's performance and power consumption. Due to the diversity in systems' performance and power consumption, a user with an application-specific performance and power budget requires suggestions for a suitable computer system(s) as shown in figure 1.1c. Therefore, it is crucial to select computer system(s) with appropriate hardware features that fit the user's budget for application performance and power, as shown in figure 1.1b.

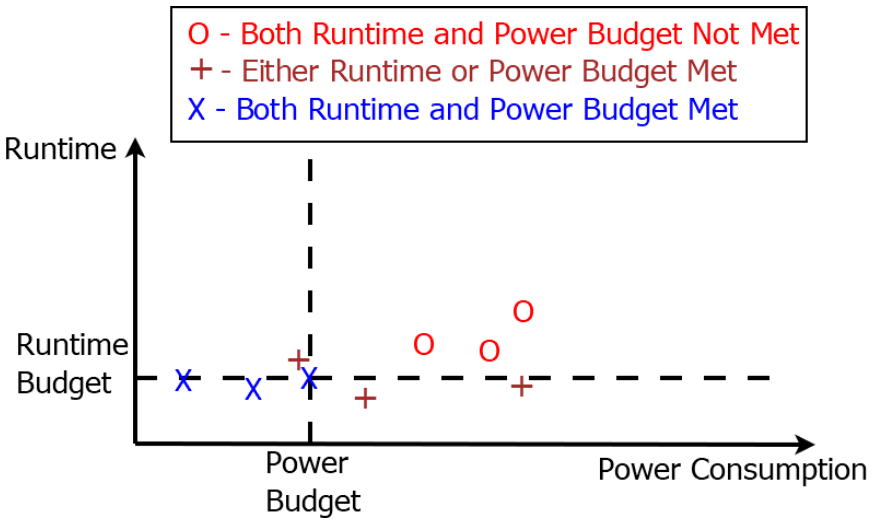
To select computer systems that conform to the user-provided budget, one requires access to physical systems in large quantities/numbers with varied hardware features to execute software applications and collect performance and power consumption data. Therefore, to collect data from a large number of physical



(a) Software Application's Performance and Power Consumption on Computer Systems with Different Hardware Features



(b) Selecting Computer System for an Application without having access to Physical Systems



(c) Computer System Selection for an Application with User Specified Runtime and Power Budget

Figure 1.1: Problem Formulation

computer systems, systems need to be procured. However, expecting a user to procure a plethora of physical systems would be prohibitively expensive. There-

fore we need a methodology that provides accurate estimates of the application's performance and power for physical computer systems without having access to them, a hard problem to solve. Over the years, academic and industry partners have collaborated to design modeling techniques to estimate the performance and power of systems with varying hardware features [3] [4] [5]. Furthermore, many researcher's have used simulators to simulate real systems [6] [7] [8] [9] [10] [11] [12] [13]. Thus, we aim to: first, construct simulated systems with real systems' features and collect approximate performance and power consumption by executing compute-intensive and data-intensive applications. Second, build models to accurately estimate the performance and power consumption of physical systems (without physically accessing them) for the approximate values collected from simulated systems. Third, we validate the accuracy of our model by comparing estimated performance and power consumption to the actual values collected from physical systems. Finally, use the estimated performance and power values to address the computer system selection problem.

1.1 Performance and Power Prediction Modeling

Performance and power modeling is an active research area due to its benefits for hardware-software co-development. System architects and system software developers use these models to improvise hardware features or applications by understanding the interactions between the system architectural features and the application performance and power [11]. Researchers have built empirical models using supervised machine learning algorithms for performance and power modeling. The first task in supervised machine learning is to build a dataset from computer systems' hardware features and the actual performance and power collected from simulated or physical systems by executing applications. The hardware features are used as input features while the actual performance and power consumption are the expected output for the machine learning model. The second task is to train the machine learning algorithm using the samples from the dataset with input features and the actual output. Once trained, the machine-learning al-

gorithm is supplied with only the input features (without the output) from samples not utilized during training to predict the output. The research community has proposed performance and power prediction models using various machine learning algorithms. The prediction accuracy of each model is determined by the type of machine learning algorithm and the dataset used for modeling. For example, the linear regression-based model in [14] [15], the tree-based model in [6] [14] [15] and the neural network-based model in [4] [14] [16] [17] each have different prediction accuracy. These research papers utilize a particular machine learning algorithm for prediction modeling. However, we evaluate widely used machine learning algorithms for their accuracy to provide accurate performance and power prediction.

We observe two possible trends in the development of performance and power prediction models using machine learning. In the first trend, a different set of applications are used to build prediction models while keeping the computer system unchanged [11] [18] [19]. In this case, the focus is to understand how types of applications, compute-intensive or data-intensive, depend on hardware features of a computer system resulting in diverse performance and power. The second trend uses a different set of computer systems having a diverse processor, cache and memory features during model training and prediction while keeping the application fixed, known as cross-platform prediction [3] [4] [5] [17]. This trend focuses on understanding how the feature changes in the processor, cache, and memory of various computer systems impact the performance and power. Our work focuses on both aspects by considering physical systems with disparate hardware features and selecting applications with different computation and data access patterns.

1.1.1 Cross Performance and Power Prediction

Cross-platform prediction models estimate the performance and power of computer systems with hardware features or instruction sets that are dissimilar to the ones used during training. A cross-platform prediction model may use hardware features either from one system only or from multiple systems during training. For example, work in [3] has proposed models to predict the performance and

power of ARM-based systems, utilizing performance counters as input features, collected during application execution on x86-based systems. Similarly, work in [5] uses two x86-based HPC systems, one as the source and the other as the target, to train the model from the features of the source system and perform predictions for the target system. The modeling work in [20] predicts a GPU's performance from features collected from an x86-based general-purpose system. On the other hand, [4] uses four systems with x86-based and ARM-based instruction-set with disparate hardware features to perform training from three systems while predicting the remaining fourth system. The work in [21] uses features of x86-based systems to predict the performance of the cloud systems.

All of these cross-platform prediction models have used physical systems for both source platforms for training and the target platforms for prediction. In contrast, we use cross-platform prediction to estimate the physical systems' performance and power from simulated systems. The work in [3] and [4] have used performance counters as input features to the machine learning model, which captures how each of the applications depends upon processor, cache or memory hardware features of the system. However, collecting performance counters requires the execution of an application on physical systems. In contrast, we use processor, cache and memory features directly to eliminate the need for physical systems. Furthermore, [3] and [4] have built two separate models, one to predict performance and the other to predict power. Alternatively, we categorize the performance and power prediction problem as a multi-target prediction due to the relationship between the performance (runtime) of the application and how much power it consumes. Therefore, we aim to build a single machine learning model to predict performance and power simultaneously for each application, called multivariate performance and power prediction models.

1.1.2 Prediction with Transfer Learning

The widespread use of machine learning models has given rise to new techniques such as transfer learning [22] for performance and power prediction. A transfer learning technique can retain the acquired knowledge during the model's training

from one task and apply the knowledge for a prediction in a subsequent new but similar task with equivalent datasets. The advantage of using the transfer learning technique for a new task is that it does not require retraining and enables the prediction with an incrementally generated dataset.

The works in [23], [24], and [5] have demonstrated that the model trained using 100% of samples from source HPC system and only 1% from target system performance data can predict 99% of the target HPC system performance data. Similarly, in [25] and [26] transfer learning is utilized to train the model in an online mode with incremental data as new data is available for improving the prediction accuracy of a given task. These transfer learning works utilize physical systems during training as well as during prediction. By contrast, we use transfer learning models to train the model from simulated systems for the prediction of physical systems.

1.2 Modeling Challenges Due To an Applications Dependence on System Features

Accurate performance predictions of a software application on a computer system is a complex task. This is due to the non-terminating and non-deterministic nature of computer systems; while the behavior of applications is terminating, deterministic and platform-independent [27] [28]. The performance of an application depends on application features, hardware features of a computer system, the runtime environment, and their mutual interactions [28]. Implementation of algorithms in the application and input problem size define application features. Processor and memory hierarchy features such as processor clock speed, the number of cores, cache features, memory size, memory access speed, memory type characterizes the computer systems. Runtime environments consist of system software tools essential to execute software applications, such as operating systems and compilers.

We define the application's performance in terms of "execution cost," which is the execution time, that is, "runtime" to complete a task on a computer system

(hardware model). Applications are characterized as compute-bound (compute-intensive), memory-bound (data-intensive) or in combination (compute-plus-memory-bound) according to the roofline-model (figure 1.2) discussed in [2] [29]. Compute-bound applications are dominated by computation (floating-point operations) rather than data access, performing multiple computations for each byte of data access. Therefore, compute-bound applications have a higher dependence on processor features. On the other hand, memory-bound applications are dominated by data access from memory rather than from computation, causing higher dependence on memory features. Although several applications are compute-bound or memory-bound, many fall into the third category (compute-plus-memory-bound) when both computation and data access operations are equally important.

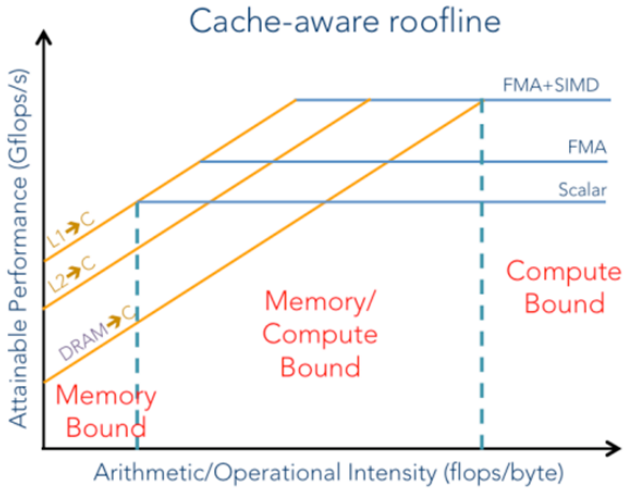


Figure 1.2: Cache-aware Roofline Model [Figure taken from PICSAR application article [1] which uses cache aware roofline model from [2]]

Performance of an application on a specific computer system's hardware is determined by a code balance (B_c) of an application and machine balance (B_m) of the computer system, which is defined as $min(1, \frac{B_m}{B_c})$. Code balance ($B_c = data\ traffic\ [bytes]/floating\ point\ ops\ [flops]$) is a good measure to characterize an application [30]. $B_c >> 1$ indicates an application is memory-bound whereas $B_c << 1$ indicates a compute-bound application. Machine balance ($B_m = Memory\ Bandwidth\ [GWords/sec]/Peak\ Performance\ [GFlops/sec]$) characterizes the computer system's hardware. A computer system hardware with higher B_m has a better bandwidth for faster data access from memory, whereas a lower B_m indicates

better processing speed. Thus, a computer system with a higher machine balance is desirable for memory-bound applications, whereas a lower machine balance system is suitable for compute-bound applications.

1.3 Motivation: Leverage Simulated Systems To Model Performance and Power for Physical Systems

To select computer system(s) that match user-provided budgets of performance and power for an application, requires access to a multitude of computer systems with diverse hardware features to execute an application and gather performance and power consumption. However, expecting a user to have physical systems as described above at their disposal is impractical. Therefore, we need a methodology to predict performance and power for physical systems without executing an application on the physical systems.

Over the years, researchers have used simulators to build systems with hardware features from real systems to gather performance and power consumption proportional to real systems. For example, PTLSim is a cycle-accurate full-system simulator with support of only x86 instruction-set indicated by [31]. SimpleScalar is primarily a processor simulator used by [6] and [7] which supports various instruction-set and processor designs. The BookSim and MultiSim simulators used by [12] and [13] mainly deal with simulating on-chip networks. However, we require a simulator that simulates out-of-order processors with multiple instruction sets (x86, ARM, etc), and multiple levels of cache hierarchy and various memory modules.

The gem5 simulator [32] has the capability of handling out-of-order processors, multi-level cache hierarchy and has a collection of memory modules. Computer systems can be built in a gem5 simulator using processor features such as clock speed, cores, memory features such as memory types (DDR2, DDR3, DDR4), memory speed, size and cache features. It also supports building systems with different instruction-set architecture (ISA) such as x86, ARM. The gem5 simulator is an open-source well-accepted cycle-accurate simulator used by academia

and industry architecture researchers [8] [9] [31] [33] [34] [35] [36] [37] to build systems. When an application is executed on gem5 simulated systems, the logs provide performance (runtime); however, collecting power for gem5 simulated systems is not a trivial matter. Work in [9] [33] [35] has shown that the McPAT (Multicore Power, Area, Timing) [38] tool used in conjunction with gem5 provides fair estimates of power consumption for applications executed on gem5 simulated systems. Hence, we leverage gem5 simulated systems logs to collect the performance (runtime) and McPAT logs to collect power consumption for application execution.

The common trend is to build simulated systems using a full system simulator (FSS). The work in [31] shows that the systems built with the full-system mode in a gem5 simulator provide close estimates of the application's performance to that of the physical systems. Additionally, performance prediction models developed using machine learning algorithms trained on the FSS performance dataset give fairly accurate predictions for systems varying in hardware features. However, constructing hundreds of computer systems with a full system simulator is arduous and slow method/process [39] [40].

On the other hand, gem5 supports what is known as system-call emulation mode, to construct computer systems rapidly. However, with the inherent design differences between the emulation mode compared to the full-system mode, the accuracy of the estimated performance and power is compromised. The focus was therefore to estimate this difference in the accuracy of performance and power, and to achieve this we built gem5 simulated systems using emulation mode based on the hardware features from available real systems. On executing the applications on these gem5 simulated systems, we found a substantial difference in performance and power estimated by the gem5 simulated systems compared to the real systems. Hence, the challenge was to develop a methodology that took advantage of both, the speed of constructing gem5 simulated systems using emulation mode combined with modeling techniques to provide accurate predictions for the physical systems.

There were two significant challenges in predicting the performance and power

of the physical systems from gem5 simulated systems built with emulation mode. The first being that the hardware features of the physical system and those simulated by the gem5 simulator may not be identical. This necessitated an estimation of performance and power for the physical system. To overcome this challenge, we explored performance and power modeling based on machine learning algorithms. We first trained the machine learning algorithm using the hardware features, performance, and power that were collected by executing an application on gem5 simulated systems. We then fed the hardware feature values identical to that of the physical system to the trained machine learning algorithm to predict the performance and power.

However, the predicted values were inaccurate estimates compared to the actual performance and power of the physical system. The inaccuracies in prediction were the outcome of a machine learning model trained only from gem5 simulated systems built using emulation mode with its inherent design differences to the full system, as stated above. Therefore, the second challenge was to improve the accuracy by bringing predicted performance and power values closer to that of the physical system's actual performance and power values. The problem compounded because the inaccuracies vary according to the application types, compute-intensive or data-intensive, due to the application's dependence on different hardware features. To overcome the second challenge, we introduced two innovative techniques based on scaling and transfer learning integrated with a machine learning model.

1.4 Thesis Problem Statement and Proposed Solutions

In this dissertation, we aim to demonstrate the possibility of combining the speed of gem5 simulated systems constructed using emulation mode with state-of-the-art modeling techniques to provide close estimates of performance and power for physical systems by solving both the challenges mentioned above. We refer to this problem as "cross performance and power prediction." An accurate prediction from the cross performance and power prediction model will facilitate the

selection of physical computer systems, by meeting the user-specified budget for the application's performance and power, without the need for executing an application on physical systems. We introduce two innovative techniques based on scaling and transfer learning to achieve this goal.

We propose a scaling technique with the assumption that access to physical systems is unavailable. We first trained a machine learning model using the performance and power dataset, collected by executing an application only on the gem5 simulated systems built with emulation mode, which we call the "Learned Model." To procure the estimates of performance and power for a physical system, we provide hardware features, identical to that of the physical system, to the learned model and to predict performance and power. However, the predicted values are coarse estimates for the physical system because the learned model is trained only from a simulated systems dataset. Therefore, the predicted values are close to the performance and power consumption of the simulated system that is built using the same physical system's hardware features used for prediction instead of actual performance and power of the physical system and thus we calculate the difference between the two. To reduce this difference and make the estimates accurate, we develop a mathematical model to derive a factor that we call scaling factor. Our mathematical model determines the scaling factor depending on application types, compute-intensive or data-intensive, by finding the correlation coefficient between the hardware features and performance or power. Finally, we applied the application-specific scaling factor to the predicted performance and power values to get accurate estimates of the performance and power values for the physical systems.

We introduced another technique based on the existing transfer learning method. In this method, we used the notion of source and target systems, both being physical systems. We considered estimating the performance and power for the target system(s) to fit in the performance and power budget provided by a user. However, our assumption for the transfer learning model was that the access to the target system was unavailable, although access to another physical system, the source system(s), was possible. We collected performance and power data along

with hardware features from the source system. If a source system has identical features to the target, it provides accurate estimates of performance and power for the target system; however, if the source system's hardware features were close to that of the target system was meant to suffice. We eventually trained the machine learning model from hardware features, performance, and power data combined from a larger gem5 simulated systems dataset and a smaller source physical system dataset. Once trained, the model predicts the performance and power for the target system, only using hardware features of the target system as input. The small percentage of source physical systems data used during model training allowed the model to improve the accuracy of performance and power for the target system. We estimated performance and power for a specific target system from multiple machine learning models, where each model was trained from a dataset of different source systems varying in hardware features. The source system with hardware features close to that of the target system provides a higher accuracy estimate than other source systems.

1.5 Thesis Contribution

The problem of selecting physical computer system(s) to meet the user-specified application-specific performance and power budget, by predicting performance and power for physical systems from the machine learning model trained only on simulated systems dataset called cross performance and power prediction, is one of its kind. This thesis presents two new techniques to address this problem. The thesis also makes several contributions to performance and power modeling research while presenting new techniques. In particular and major contributions are:

- To device a solution for cross prediction, it is important to understand how performance varies between physical systems compared to simulated systems even when both the types of systems have an identical processor, cache and memory features. Therefore, we first introduced a "learning-based" model to train and predict from within the same type of systems, both phys-

ical and simulated. The learning-based model trained from physical or simulated systems' data would be able to provide physical or simulated system performance given the hardware features. Our learning-based model has three phases of data collection, training and prediction. In data collection, to build simulated systems' datasets, we first constructed 475 simulated systems in the gem5 simulator using the system-call emulation mode. This was done utilizing real systems hardware features like the type of processors (instruction-set, cores, speed and cache configuration), type of memory, its speed and size. We executed compute-intensive and data-intensive applications on these simulated systems to collect application-specific performance from each simulated system. We assembled the known hardware features and application-specific performance to build the application-specific simulated systems dataset. To build datasets for physical systems, ten available physical systems were selected with disparate hardware features and we collected hardware feature values using the dmidecode utility. We executed the same set of applications on the selected physical systems and measured the performance. Application-specific performance datasets are built by assembling physical systems' hardware features and performance. The training and prediction phases were performed using two machine learning algorithms, linear regression (LR) and decision tree regression (DTR). In the training phase, a set of LR and DTR machine learning algorithms were trained from 80% samples from the application-specific simulated or physical systems datasets using hardware features and the actual performance, while the prediction performed on the remaining 20% data was done using only hardware features as input. The results show that due to the non-deterministic nature of the physical systems, the model accuracy for physical systems was lower than the model accuracy for simulated systems. Furthermore, due to the high manufacturer variability of processors compared to memory modules in physical systems, compute-intensive applications have lower accuracy compared to data-intensive applications. It was also observed that the DTR algorithm had higher prediction accuracy compared

to LR in both models for simulated as well as physical systems, due to the non-linear relationship between hardware features and performance that is captured by DTR [41].

- We have evaluated two machine learning algorithms LR and DTR, in our first work. However, researchers have used many machine learning algorithms to build performance prediction models. Linear, tree-based, Gaussian, support vector, and neural networks are classes of machine learning algorithms. The prediction accuracy of prediction models varies depending on which class of machine learning algorithm is used as well as the characteristics of the performance dataset. Hence, it is essential to evaluate machine learning algorithms from these classes to find out which one provides the highest prediction accuracy. In [42] [43], we assess fourteen machine learning algorithms to identify the one that provides better prediction accuracy for various benchmark applications. The result shows that the tree-based machine learning algorithm class, including the decision tree regression (DTR) algorithm, provides a higher prediction accuracy compared to the other machine learning algorithms classes. Due to its simplicity and intelligibility, the decision tree algorithm is the most straightforward tree-based machine learning algorithm. DTR builds a binary tree with each non-leaf node with a rule (condition) during the training phase, which gets evaluated during the prediction phase to reach the leaf node with a predicted value. Our understanding is that the binary tree data structure makes the decision tree algorithm learn the non-linear relationship between hardware features and performance. To validate this understanding, we present a process in [44] to extract the set of rules from a binary tree of trained decision tree model tracing predicted values from the given hardware features as an input.

- We have developed prediction models only for performance (runtime) up till now. However, our aim is to build prediction models for both performance and power. Therefore, we extend our learning-based model further in [45] to include a necessary attribute that of power in addition to the per-

formance, which we refer to as multivariate performance and power prediction model. Machine learning algorithms such as linear regression (LR) can predict only one value, therefore, it requires two models to predict performance and power individually. However, algorithms such as decision tree regression (DTR) predict performance and power simultaneously using only one instance of an algorithm. We use the DTR algorithm to develop multivariate performance and power prediction models because there is a relationship between an application's performance and how much power it consumes. For multivariate performance and power modeling, we need to first collect power consumption for each application executed on each simulation or a physical system which is non-trivial. For the simulated systems dataset, we leverage the McPAT tool in conjunction with gem5 to collect power consumption. We then assembled the system configurations and execution statistics generated from all 475 gem5 simulated systems and transformed the information into a format acceptable to McPAT. We built a simulated systems performance and power dataset from hardware features, performance collected from gem5 logs, and power consumption collected from McPAT for each application execution on every gem5 simulated system. The decision tree model trained from simulated systems' performance and power dataset, simultaneously predicts performance and power with an accuracy of 95% for applications used in experiments. To collect power consumption for each application execution on the ten physical systems, we leveraged the PAPI (Performance Application Programming Interface) toolset. Our work in [46] shows that with some modification to the application code, we can invoke the PAPI (API) (Application Programming Interface) to collect the power consumption.

- All prediction models built so far are trained from either simulated systems datasets or physical systems datasets to perform predictions for the same type of systems (i.e. either simulated or physical). We aim to on the other hand build prediction models to predict performance and power for physical systems from a model trained on simulated systems dataset,

called the cross performance and power prediction model. We introduced our first cross prediction model based on a scaling technique in [47] to predict the performance of the physical system from a model trained using the gem5 simulated systems performance dataset. The cross performance prediction model was further extended in [48] to include power predictions called the "Cross Performance and Power Prediction Model with Scaling." To implement a cross prediction model with scaling, we first built application-specific machine learning models by training a decision tree algorithm on simulated systems performance and power datasets for six benchmark applications from the San Diego Vision Benchmark Suite (SD-VBS) and MiBench. We then provided hardware feature values identical to that of the physical system as an input to the trained model to predict the performance and power of the physical system. There are dissimilarities between the gem5 simulated systems (built with emulation mode) and physical systems. Due to these inherent dissimilarities and compounded by the fact that the machine learning model is trained only from the gem5 simulated systems dataset, the predicted values therefore from the model are analogous to the performance and power of simulated systems. But the predicted values differ by a factor from the actual values of physical systems. We calculated this factor by the scaling technique, which we label as the "Scaling Factor." The scaling factor has two components: a major factor that captures the dissimilarity in design between the gem5 simulated systems and physical systems, and a minor factor is a deviation in the major factor which is contributed by the variance in performance and power due to compute-intensive or data-intensive applications and their dependence on different hardware features. To assess the major factor, we built gem5 simulated systems using the hardware feature values from the physical systems and collected the performance and power by executing the same six benchmark applications on these gem5 simulated systems. The mean difference between the performance and power of these simulated systems and the physical systems is the major factor. To derive the minor factors which are the variations in

the major factor caused by applications' dependence on different hardware features, we used the correlation coefficient between hardware features and performance or power for each benchmark application. Using the major and minor factors, we calculated the application-specific scaling factor and applied it to the predicted performance and power for accurate physical systems performance and power predictions for a given application. The result demonstrates the prediction accuracy to be between 75%-90% for performance and 60%-95% for power for all the benchmark applications.

- Our work in [46] [49] introduces two cross prediction models based on transfer learning techniques, cross-platform predictions and cross-systems predictions. In the cross-platform prediction, we used alternative instruction sets (x86 or ARM) or system architectures (Intel Core or Intel Xeon) as the platforms, in which one platform (training platform) is used for training, the other for predictions (prediction platform). On the other hand, in cross-systems predictions, we train the models from simulated systems (training systems) dataset and predictions are performed for physical systems (prediction systems). However, predictions could have lower accuracy due to vastly different measurement ranges of performance and power for the dissimilar platforms or system types used during training and prediction. Therefore, models are trained using 100% training platforms or systems datasets and 1-10% prediction platforms or systems datasets to predict the remaining 99-90% of performance and power of the prediction platforms or systems dataset. The cross-platform and cross-systems prediction works in [46] [49] also evaluates univariate machine learning algorithms such as linear regression, nearest neighbor, gaussian process regressor and multivariate machine learning algorithms such as decision tree, random forest, and neural network. The result further confirmed that the tree-based models outperform the other models. We extended the cross-systems prediction model based on transfer learning using the decision tree algorithm in [44] called the "Cross Performance and Power Prediction Model with Transfer Learning." In this model, we used physical systems with the notion of

source and target systems. The assumption for the transfer learning model was that access to source systems was possible, while access to target systems was unavailable and therefore target systems require prediction. The transfer learning model uses a pair of source and target systems for training and prediction. We train the model from the gem5 simulated systems dataset combined with a source system dataset to predict the performance and power of the target system. We predict performance and power for the same target multiple times, each time a transfer learning model is trained using an alternative source physical system dataset. The result demonstrated that the source system having hardware features close to that of the target system provided higher prediction accuracy. We selected the source system that provided the highest accuracy for each target system for the final result. The result of the transfer learning model achieved the mean prediction accuracy of 50% for all benchmark applications.

1.6 Thesis Organization

The rest of the thesis is organized as follows:

Chapter 2 introduces the learning-based model based on machine learning for performance modeling with three phases, data collection, training and prediction. In the data collection phase, first, the work presents the process of constructing gem5 simulated systems using emulation mode along with the challenges faced and the solutions implemented to resolve them. It further provides a selection of physical systems and compute-intensive and data-intensive applications to build performance datasets for simulated and physical systems. For the training and prediction phases, the work uses these datasets to build machine learning models using linear regression (LR) and decision tree regression (DTR) algorithms. The work builds separate models for simulated systems datasets and physical systems datasets for each application to show how accuracy for the same application deviates between simulated and physical systems.

Chapter 3 evaluates the accuracy of performance prediction models built using

14 machine learning algorithms. The work develops models using ten algorithms from the scikit-learn library such as support vector, multiple linear regression, ridge regression, k-nearest neighbor, gaussian process, decision tree, random forest, extremely randomized trees, gradient boosting, and extreme gradient boosting. The remaining four are variants of a neural network built using the Keras library. The work then provides a comparative analysis of the prediction accuracy from all 14 algorithms for eight compute-intensive and data-intensive applications' performance datasets built from simulated and physical systems.

Chapter 4 introduces a multivariate machine learning model using a decision tree algorithm to perform predictions of performance and power simultaneously. In this work, the multivariate model is evaluated for the simulated systems performance and power dataset requiring assimilation of power consumption data from gem5 simulated systems. Therefore, the work presents a process to compile the required information from gem5 simulated systems upon application execution that is fed to the McPAT tool to compute dynamic power consumption. The work then evaluates the prediction accuracy of multivariate performance and power models for seven compute-intensive and data-intensive benchmark applications.

Chapter 5 introduces the cross performance and power prediction model with scaling. The cross prediction model with scaling uses a scaling factor consisting of two factors, major and minor, to predict physical systems performance and power from the gem5 simulated systems dataset. The work presents the derivation of both major and minor factors. Furthermore, the use of correlation coefficient to determine application-specific minor factors is also presented. The work evaluates the model accuracy for six compute-intensive and data-intensive benchmark applications.

Chapter 6 introduces the transfer learning technique for cross prediction. This work performs cross-systems prediction and cross-platform prediction using transfer learning. In cross-systems prediction, the work predicts performance and power for the physical system using a machine learning model trained from simulated systems' dataset. On the other hand, in cross-platform prediction, the work

trains the model using x86-based systems data to predict performance and power for ARM-based systems in simulated systems. Similarly, for physical systems, a trained model from Intel Core-based systems data predicts Intel Xeon-based systems data.

Chapter 7 first extends the transfer learning model from chapter 6 to train a machine learning model from a source physical system dataset coupled with a simulated systems dataset for the prediction of the target physical system. The work shows that when a machine learning is trained from a dataset of a source physical system whose hardware features are close to that of the target physical system, the transfer learning model provides higher prediction accuracy. Finally, the work carries out a comparative analysis of both the cross performance prediction models, scaling and transfer learning.

Chapter 8 summarizes the contributions with conclusions and outlines future work.

CHAPTER 2

Learning Based Performance Prediction of Applications on Disparate Computer Systems

2.1 Overview

Today, many options of computer systems are available from commercially of-the-self (COTS) manufacturers or from the cloud to host the software. These computer systems primarily vary in terms of memory and processor features. Software execution on dissimilar systems with diverse feature values has different performances and hardware costs. It is, therefore, crucial to select a system that provides optimum performance for a given software within the requisite budget. This would require an accurate performance prediction of the software on a given system without actually running the software. In this chapter, we aim to propose a learning-based performance prediction model for hardware configuration selection that would be best suited for the software execution time defined by a client for a given software.

Performance prediction has always been a very important research area. Our area of interest is performance prediction in multicore systems that are widely used systems today. Some research works [20] [50] [51] have focused on cross-platform performance prediction to predict the performance of an unknown application by collecting features on one architecture while predicting the performance for the different target architecture. A cross-platform performance prediction of an unseen software for ARM-based target by collecting performance counters on an x86-based system is reported in [50]. Similarly, a cross-platform predic-

tion of unknown software for GPU, by collecting application features (ILP, memory reference with reuse distance of two, single versus double-precision floating-point operations, integer operations, etc.) on a general-purpose x86-based processor is reported in [20] with an average prediction error of 26.9%. Speedup prediction for OpenMP applications on a GPU is performed by collecting memory, computation, control-flow, etc., features from general-purpose processors are reported in [51] with prediction accuracy of 77% to 90%. These research works focus on predicting the performance of new software for the same target system that was used during training. On the other hand, our motivation is to predict the performance of the same software on new target systems unseen during training to select the one with optimal performance.

Numerous research efforts have focused on specific hardware domains such as GPU [15] [40] [18], cloud [21] [52], or HPC systems [5]. The effect on the performance of ninety-seven OpenCL applications, by scaling compute units (CU) from 4 to 44 with a frequency ranging from 200MHz to one GHz and GDDR5 memory bandwidth from 38.4 to 320 GB/s by changing frequency in the range of 150 to 1250 MHz for AMD FirePro™ W9100 GPU has been reported in [40] and [18] with a performance prediction error of 10% to 15% for new scaling factors. Work in [21] proposes a performance prediction model that predicts the application's performance on a cloud platform before deployment by the amalgamation of features from the application's profile on a non-cloud based system collected using a platform-independent software analysis (PISA) tool and cloud platform features collected as cloud profile. The novel idea of rule-based auto-scaling, using the probabilistic model, in support of the QoS performance guarantee, in the cloud environment of an application, is proposed in [52]. Performance prediction of mini-applications from the Mantevo suite on four large-scale HPC systems (two IBM Blue Gene/Q and two Cray systems) has been reported in [5] with a miniFE application with a performance prediction error of about 10%.

Many research efforts [14] [17] [53] [54] [55] have focused on utilizing performance prediction to understand and improve upon certain multicore systems characteristics. For example, [53] proposes a linear regression-based model for

fine-grain profiling prediction of MPI applications by code block categorization method with median prediction error of 8% to 13% on TiahHe-2 and Taub clusters considering two NAS Parallel Benchmarks (NPB). The effect of shared resources contention upon execution of multiple software applications on multicore architecture using performance prediction is the focus of [14] and [54]. Improvement in a thread scheduling or task scheduling has been reported in [55] by predicting the performance of a task to core assignment in the multicore system.

Performance prediction is applied to study the effect on software performance based on the evolution of different computer systems in [12] [13] [16] [17]. Performance prediction of SPEC CPU benchmarks is performed by building an artificial neural network (ANN) model that uses relevant features (System, number of Cores, number of Chips, number of Cores Per Chip, number of Threads Per Core, Processor clock-speed, Level 1 Cache, Level 2 Cache, Last Level Cache, Memory, and Compiler) selected using principal component analysis (PCA) with an accuracy of 97.5% is reported in [17]. In [12] the effect of different network topologies are studied for on-chip networks considering different traffic patterns using the BookSim simulator and the reported prediction error of 8%. Online prediction of performance in terms of the future processor cycles is performed in [16] by considering nine processor features for models built in the M-sim simulator and collecting 18 execution parameters at a specific cycle interval with maximum prediction accuracy of 70%. Design space exploration is the main goal in [13] to develop different systems by predicting system parameter values using linear regression, support vector machine regression and k-nearest-neighbor regression.

We have three important observations from literature survey on performance prediction research summarized as: (1) Most of the prediction models were built for a specific type of instruction-set-architecture; either ARM-based or x86-based but not both. (2) Performance prediction models were built either for simulated systems [12] [16] or for physical systems [5] [14] [55]. (3) Many of the performance prediction research efforts used standard benchmark applications; for example, SPEC benchmark used in [16] [17] [54] [55], NPB in [14] [53], PARSEC in [13]; thereby treating the application as a black-box.

In this work, we proposed a machine learning-based models using linear regression (LR) [7] [56] and decision tree regression (DTR) [57] for performance prediction of applications on different multicore computer systems. The four important attributes of this work are: Firstly, we have built a single performance prediction model to perform predictions for multiple instruction-set-architecture (ISA), ARM-based and x86-based. Secondly, our performance prediction framework consists of both simulated systems built using the gem5 [32] simulator and physical systems for training as well as prediction. Thirdly, we have selected three applications according to their known compute-intensive (compute-bound) and data-intensive (memory-bound) patterns to consider them as white-box. Finally, to establish the generality of the model, we have tested the performance prediction model on EP (compute-bound) and MG (memory-bound) benchmarks from NAS Parallel Benchmarks (NPB) [58] and miniFE from the Mantevo [59] suite, mini-applications, representing real-world applications with multiple phases.

The performance prediction model results could be used to create performance-based clusters of hardware systems, in which hardware systems with similar performance will make one cluster to enable hardware selection based on the application's performance on each of the clusters.

The remainder of the chapter is organized as follows: Section 2.2 describes a learning-based performance prediction model that we proposed in this chapter. We have used two machine learning models for the performance prediction of three applications on simulated as well as real hardware systems. Section 2.3 provides information regarding dataset construction by describing the procedure to build computer systems in the simulator, selection of applications and generating datasets by executing these applications on simulated and physical systems. Section 2.4 provides insight into performing data normalization and handling both categorical and real-valued features. Section 2.5 describes learning-based models and how they are used to learn the relationship between the system's hardware features and the application's runtime. Section 2.6 articulates the results from learning-based models from both simulated as well as physical systems. Additionally, it provides an analysis of the result to provide insight into model selec-

tion. Section 2.7 discusses threats to internal and external validity as well as the limitations. Finally, section 2.8 has concluding remarks along with tasks that we plan to continue.

2.2 Learning Based Performance Prediction Model

Accurate performance prediction of an application on a computer system is a complex task because systems are in general non-terminating and non-deterministic, whereas the behavior of applications is terminating, deterministic and platform-independent [27] [28]. The performance of an application (software) depends on a wide range of factors like the nature of the application, the hardware features of the computer system, the runtime environment, the input (problem size), the measurement methodology, etc. and their mutual interaction [28]. Computer systems can be characterized by processor features, memory hierarchy features, and network features.

We define the application's performance in terms of "execution cost," which is the execution time, that is, "runtime" to complete a task on a computer system. Applications are characterized as compute-bound, memory-bound or compute-plus-memory-bound according to the roofline-model discussed in [2] [29]. Compute-bound applications are dominated by computation (floating-point operations) rather than data access, which means, multiple computations are performed on each byte of data that is accessed. On the other hand, memory-bound applications are dominated by data access from the memory rather than computation. Although several applications are compute-bound or memory-bound, many fall into the third category of compute-plus-memory-bound, in which both computation and data access operations are equally important. Application's performance on a specific computer system is determined by application's code balance (B_c) and computer system's machine balance (B_m) which is defined as $min(1, \frac{B_m}{B_c})$. Code balance ($B_c = data\ traffic\ [bytes] / floating\ point\ ops\ [flops]$) is a good measure to characterize an application [30]. $B_c >> 1$ indicates an application is memory-bound whereas $B_c << 1$ indicates it is compute-bound. Machine bal-

ance ($B_m = Memory\ Bandwidth$
$[GWords/sec]/Peak\ Performance\ [GFlops/sec]$) characterizes a computer system; higher B_m indicates that the system has better memory data access speed whereas lower B_m indicates better processing speed. Therefore, a computer system with higher machine balance is more suited for memory-bound applications, whereas a system with a lower machine balance is a good fit for compute-bound applications.

Intuitively, we know that there is a relationship between the machine balance of computer systems and their features. For example, memory access speed (bandwidth) will be determined by memory features such as memory clock speed, memory size, memory type, etc. And the peak computation performance will be determined by processor features such as processor clock speed, number of processors, instruction set architecture (ISA), and so on. Therefore, the performance of a compute-bound (compute-intensive) application is dictated primarily by processor features, whereas memory features dictate the performance of a memory-bound (data-intensive) application. We leverage this understanding to employ a learning-based performance prediction model to learn the relationship between computer systems' hardware features and application performance. Our approach for the learning-based performance prediction model is shown in figure 2.1.

Our learning-based performance prediction model has three phases; data collection phase, learning (training) phase, and prediction (testing) phase. In the data collection phase, we executed the chosen set of applications (say matrix multiplication, image processing, etc.) on selected computer systems and gathered the system's hardware features and the actual runtime as shown in figure 2.1 to build an application-specific performance dataset. The performance dataset samples were divided into training samples and testing samples using the train-test split ratio. In the training phase, the learning model was trained using training samples for each application to learn the relationship between the system's hardware features and the actual performance ("actual runtime") of the same application. We selected training samples (M) from performance dataset for the training

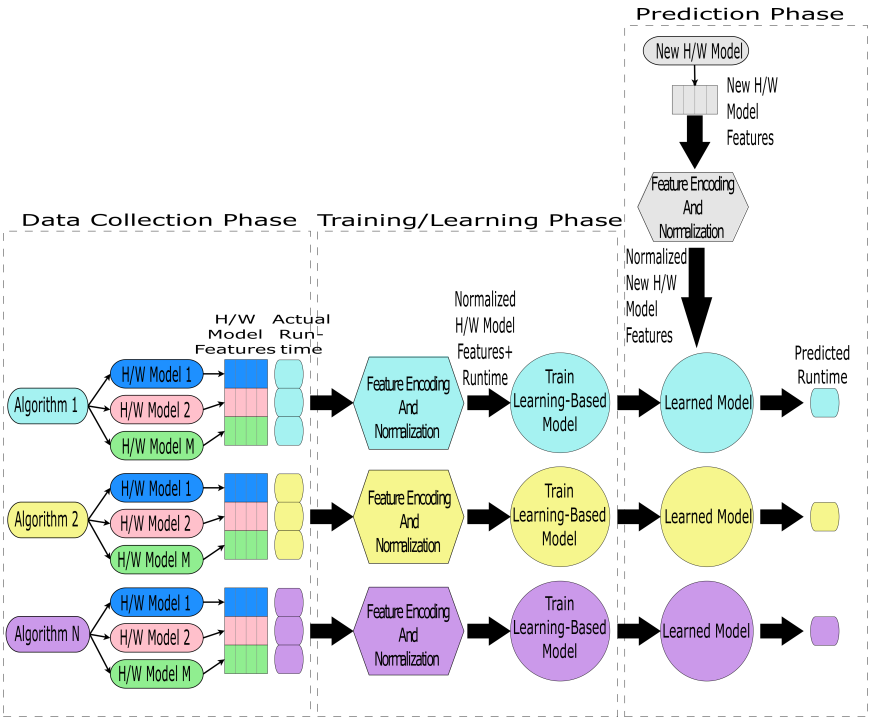


Figure 2.1: Data Driven Learning-Based Performance Prediction Model

phase with feature values that represents the general population of systems available today. We denote X_j as a set of system features with categorical or continuous values for system $j, 1 \leq j \leq M$. Categorical feature values have ordered integers or textual data, whereas continuous feature values have real or floating-point values. We encoded categorical valued features and normalized complete feature set X_j into $X'_j \in \mathbb{R}^d$ as explained in section 2.4 to improve the performance prediction model training. d in $\mathbb{R}^d$ represents the dimension of real-valued feature space X_j. For example, if we use nine system features, it will be represented as $\mathbb{R}^9$. We used the actual runtime of selected (M) samples denoted as $y_j \in \mathbb{R}$ in the training phase to train the model. The objective of the training phase is to find a function $\Im(X'_j) \approx y_j \forall j$ for which we employ machine learning regression models. At the end of the training phase, the machine learning regression model learns function $\Im$, the mapping between normalized system features and actual runtime of an application, which we refer to as "Learned Model." A different implementation of the same application (say matrix multiplication) may result in different runtimes range; hence for this model different implementation is to be treated as a different application and will need to be trained separately.

In the prediction phase, we selected a new set of systems (testing samples) unseen during the training phase, and therefore actual runtimes of a given application on these systems were unknown. We gathered features of these new systems denoted as X_{new} and followed the same normalization process to get $X'_{new} \in \mathbb{R}^d$. Normalized system features from new systems X'_{new} were provided as an input to the learned model to predict the runtime y_{new} of a given application on new systems using learned relationship from training phase $\Im(X'_{new}) \rightarrow y_{new}$.

2.3 Dataset Construction

2.3.1 Construction of Simulation-based Systems in Gem5 Simulator

Several research works have utilized various simulators to simulate real systems. For example, [6] and [7] uses SimpleScalar, [13] uses MultiSim, [12] uses BookSim. However, due to the advantages of gem5 many recent researchers [8] [9] [31] [33] [34] [35] [36] [37] have utilized gem5 simulator. We primarily use the gem5 simulator due to its following advantages:

- The gem5 simulator has the system-call emulation mode to simulate systems rapidly. Therefore, we use system-call emulation mode instead of full-system mode to build simulation-based systems in the gem5 simulator.
- The gem5 simulator supports processors with six different instruction-set-architectures (ISAs), including ARM and x86.
- The gem5 simulator supports various processor types, including the out-of-order (OoO) widely used in real computer systems.
- The gem5 simulator can be used in conjunction with McPAT to collect power consumption for applications executed on Gem5 simulated systems.

In this section, we provide details of building the gem5 simulated systems using system-call emulation mode which we used to build the simulated systems

performance dataset. We also articulate the challenges faced and changes made to the gem5 source code to overcome these challenges.

2.3.1.1 Selection of Systems

The gem5 simulated systems must represent the general population of physical computer systems available in the market today. Furthermore, gem5 simulated systems need to be characterized by their processor, cache and memory features. Therefore, we surveyed a wide range of commercial computer systems and categorized them into different hardware classes represented by the "H/W class" column in table 2.1. The footnote shows the H/W class values and its class description. For systems of each H/W class, we collected values of nine features; three processor features, CPU speed, instruction-set-architecture (ISA), cores, three-level cache sizes, and three memory features, type, access speed and size. For example, class 1 systems were build using systems features of AMD Ryzen and EPYC, class 5 systems were built using Apple systems features, class 6 systems were built based on Intel Core and so on. Table 2.1 shows feature values that we have used to build gem5 simulated systems.

Table 2.1: Computer Systems 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	32kB-64MB	50
2	x86	2.8-4.7	1-8	DDR3	1600-1866	16kB-8MB	60
3	ARM	1.7-2	4,8	DDR4	1866	32kb-8MB	15
4	ARM	1-2.7	2-8	LPDDR2	400-1866	4kB-3MB	70
5	ARM	1.1-2.34	1-4	LPDDR3	1600-1866	32kB-4MB	35
6	x86	1.3-3.5	2,4	DDR3	1600	32kB-8MB	60
7	x86	1.7-3.5	2-18	DDR4	1866-266	32kB-16MB	95
8	x86	1.3-3.5	2,4	LPDDR3	1600-2133	32kB-8MB	90

*Memory Size range 1GB to 8GB
H/W Class and its associated class of systems
1. AMD Ryzen and Epyc. 2. AMD Bulldozer and Piledriver. 3. AMD Opteron 4. Qualcomm Snapdragon. 5. Apple. 6. Intel Core i7, i5 and i3 with DDR3 DRAM. 7. Intel Core i9, i7, i5 and i3 with DDR4 DRAM. 8. Intel Core i7, i5 and i3 with LPDDR3 DRAM

2.3.1.2 Challenges During Construction of Simulated Systems in Gem5

Due to the support of several instruction set architectures (ISAs) in gem5, we built 120 ARM-based systems and 355 x86-based systems in gem5, with a total of 475 systems as shown by the "cnt" column of table 2.1. However, we faced several challenges while using other hardware features such as types of memory, three levels of cache and so on for building systems in the gem5 simulator. In the subsections below, we discuss each challenge and the solution we have applied to resolve it. We also discuss the limitations of gem5 simulated systems compared to the physical systems.

2.3.1.2.1 Challenge 1: Gem5 Memory Support

The gem5 simulator supports several memory modules with a diversity of memory types and access speed as shown in table 2.2. However, the challenge is that the gem5 simulator does not support all the required memory modules with different memory types and access speeds to build simulation systems in the gem5 simulator that represent the surveyed commercial computer systems. To overcome this challenge, we reviewed the gem5 source code available with open-access and made changes to add all the required memory modules.

Table 2.2: Supported Memory Modules in Gem5 Simulator

Mem Type	Mem Access MHz	Based On Datasheet
DDR3	1600, 2100	Micron MT41J512M8
DDR4	2400	Micron MT40A512M16
LPDDR2	1066	Micron MT42L128M32D1
LPDDR3	1600	Micron EDF8132A1MC
GDDR5	4000	SK Hynix H5GQ1H24AFR
HMC	2500	
WideIO	200	
HMB	1000	

Table 2.3 shows all the memory modules with respective memory types and access speeds added in the gem5 simulator.

The code below shows an example of the changes made to DRAMCtrl.py python code in gem5 source to add two of the memory modules. It is evident from

Table 2.3: Memory Modules Added in Gem5 Simulator

Mem Type	Mem Access MHz	Based On Datasheet
DDR3	1066, 1333, 1866	Micron MT41J512M8
DDR4	1866, 2133, 2666	Micron MT40A512M16
LPDDR2	933, 800, 667, 533, 400, 333	Micron MT42L128M32D1
LPDDR3	1866	Micron EDFA164A1PB
LPDDR3	2133	Micron EDFA232A1MA

the code example that we needed to use specification values from the datasheet of memory provided by the manufacturer to add memory modules in gem5. The "Based On Datasheet" column in table 2.3 indicates the datasheets we have used to collect the specification values.

```
577 class DDR3_1866_x64(DDR3_1600_x64):
578     # 933 MHz
579     tCK = '1.07ns'
580
581     # 8 beats across an x64 interface translates to 4 clocks @ 933 MHz
582     tBURST = '4.28ns'
583
584     # DDR3-1866 13-13-13
585     tRCD = '13.91ns'
586     tCL = '13.91ns'
587     tRP = '13.91ns'
588     tRAS = '34ns'
589     tRRD = '5ns'
590     tXAW = '27ns'
591
592     # Current values from datasheet Die Rev E,J
593     IDD0 = '62mA'
594     IDD2N = '35mA'
595     IDD3N = '41mA'
596     IDD4W = '141mA'
597     IDD4R = '174mA'
598     IDD5 = '242mA'
599     IDD3P1 = '41mA'
600     IDD2P1 = '37mA'
```

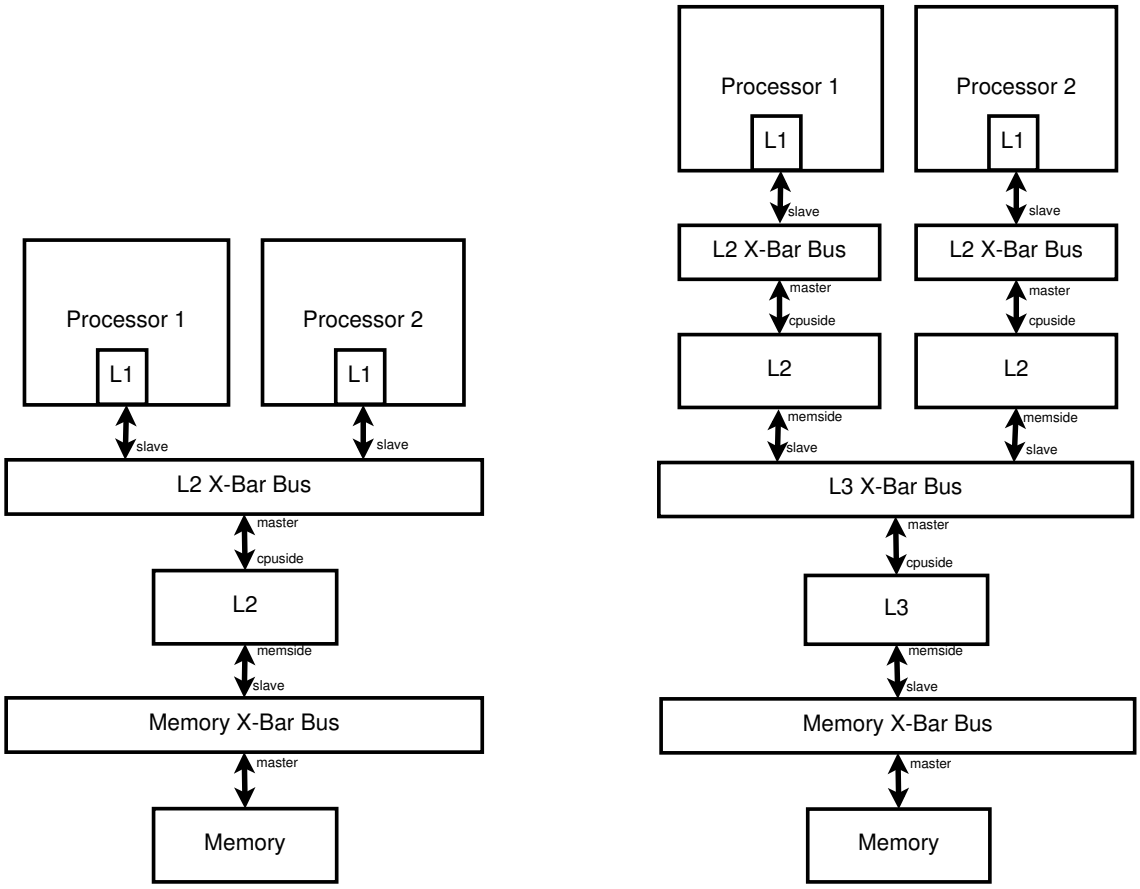
```
    IDD6 ='20mA'
    VDD = '1.5V'
```

```
class LPDDR2_933_x32(LPDDR2_S4_1066_x32):
    # 466.5 MHz
    tCK = '2.144ns'

    # Irrespective of speed grade, tWTR is 7.5 ns
    tWTR = '7.5ns'

    # Default same rank rd-to-wr bus turnaround to 2 CK, @466.5 MHz =
    2.144 ns
    tRTW = '4.288ns'

    # Default different rank bus delay to 2 CK, @466.5 MHz = 2.144 ns
    tCS = '4.288ns'

    # Current values from datasheet
    IDD4W2 = '185mA'  # 190 - 5
    IDD4R2 = '194mA'  # 220 - 194 = 26
```

2.3.1.2.2 Challenge 2: Gem5 Cache Support

The gem5 simulator supports a two-level cache hierarchy for systems building as shown in figure 2.2a, whereas most commercial systems used today have a three-level cache, a second challenge we faced in building simulated systems in gem5. To cope with this challenge, we analyzed the source code that implements the cache structure in gem5 and identified two problems that require resolution.

First, the current implementation of gem5 supports only two levels of cache (L1 and L2); hence, we made code changes to add the third level cache L3. The new L3 cache being a last-level cache in the new cache configuration will be shared between all the processors, making it a system-level cache. However, in the current implementation, the L2 cache is a last-level cache; therefore L2 is a system-level cache shared by all processors, which was a second problem. Hence, we added an option to have the L2 cache shared by all processors to make it a system-level cache or individual L2 cache for each processor. Figure 2.2b shows



(a) Gem5 Simulator Existing Two-Level Cache Configuration

(b) Gem5 Simulator Cache Configuration Modified to Three-Level

Figure 2.2: Gem5 Simulator System Cache Configuration

the three-level cache configuration that we implemented in gem5.

To implement the three-level cache configuration, we first needed to support system-level (shared) L2 cache and individual processor (non-shared) L2 cache modes to ensure that backward compatibility with the two-level cache configuration. We used an option l2cache_sharedbycpu with true or false values depending on whether the two-level configuration required a shared L2 cache or a non-shared L2 cache in a three-level configuration. The code listing below articulates the changes made to CacheConfig.py to check for the l2cache_sharedbycpu option and create either system-level shared L2 cache or non-shared L2 cache for each processor.

```
114     elif options.l2cache:
115 # 22-Mar-2018 - AKM - Added Shared L2 Cache
116         # if l2cache_sharedbycpu true then l2 cache is as cpu level and
```

```
    not system wide
        if not options.l2cache_sharedbycpu:
# 22-Mar-2018 - AKM - Added Shared L2 Cache
            # Provide a clock for the L2 and the L1-to-L2 bus here as
    they
            # are not connected using addTwoLevelCacheHierarchy. Use
    the
            # same clock as the CPUs.
            system.l2 = l2_cache_class(clk_domain=system.cpu_clk_domain
    ,
                                       size=options.l2_size,
                                       assoc=options.l2_assoc)

            system.tol2bus = L2XBar(clk_domain = system.cpu_clk_domain)
            system.l2.cpu_side = system.tol2bus.master
            system.l2.mem_side = system.membus.slave
```

```
# 22-Mar-2018 - AKM - Added Shared L2 Cache
# if l2cache_sharedbycpu true then l2 cache is as cpu level and not
    system wide
if options.l2cache_sharedbycpu:
    l2 = L2Cache(clk_domain=system.cpu_clk_domain,
                                    size=options.l2_size,
                                    assoc=options.l2_assoc)

    l2.writeback_clean = True
    system.cpu[i].addTwoLevelCacheHierarchy(icache, dcache, l2,
                                        iwalkcache, dwalkcache)
else:
# 22-Mar-2018 - AKM - Added Shared L2 Cache
```

To add L3 cache to complete the three-level cache configuration, we first added an option for L3 cache in Options.py.

```
# 21-Mar-2018 - AKM - Added L3 Cache
    parser.add_option("--l2cache-sharedbycpu", action="store_true") #
    --l2cache option must be true to use this option
    parser.add_option("--l3cache", action="store_true")
# 21-Mar-2018 - AKM - Added L3 Cache
```

We then added L3 cache in Caches.py and cross bar switch (Xbar) in XBar.py that will connect L3 cache to the memory.

```
79 # 21-Mar-2018 - AKM - Added L3 Cache
80 class L3Cache(Cache):
81     assoc = 12
82     tag_latency = 36
83     data_latency = 36
84     response_latency = 36
85     mshrs = 36
86     tgts_per_mshr = 12
87     write_buffers = 8
88 # 21-Mar-2018 - AKM - Added L3 Cache
```

```
142 # 21-Mar-2018 - AKM - Added L3 Cache
143 # We use a coherent crossbar to connect multiple masters to the L3
144 # caches. Normally this crossbar would be part of the cache itself.
145 class L3XBar(CoherentXBar):
146     # 256-bit crossbar by default
147     width = 32
148
149     # Assume that most of this is covered by the cache latencies, with
150     # no more than a single pipeline stage for any packet.
151     frontend_latency = 1
152     forward_latency = 0
153     response_latency = 1
154     snoop_response_latency = 1
155
156     # Use a snoop-filter by default, and set the latency to zero as
157     # the lookup is assumed to overlap with the frontend latency of
158     # the crossbar
159     snoop_filter = SnoopFilter(lookup_latency = 0)
160 # 21-Mar-2018 - AKM - Added L3 Cache
```

Finally, we connected L3 cache to L2 cache on one side and memory on the other side using the cross bar bus.

```
83 # 21-Mar-2018 - AKM - Added L3 Cache
84     if options.l3cache and options.l2cache:
85         print "Creating System wide L3 Cache"
```

```
86          # Provide a clock for the L3 and the L2-to-L3 bus here as they
87          # are not connected using addTwoLevelCacheHierarchy. Use the
88          # same clock as the CPUs.
89          system.l3 = L3Cache(clk_domain=system.cpu_clk_domain,
90                              size=options.l3_size,
91                              assoc=options.l3_assoc)
92
93          system.l3.writeback_clean = True
94          system.tol3bus = L3XBar(clk_domain = system.cpu_clk_domain)
95          system.l3.cpu_side = system.tol3bus.master
96          system.l3.mem_side = system.membus.slave
97
98          # if l2cache_sharedbycpu true then l2 cache is as cpu level and
     not system wide
99          if not options.l2cache_sharedbycpu:
100             print "Creating System wide L2 Cache"
101             # Provide a clock for the L2 and the L1-to-L2 bus here as
    they
102             # are not connected using addTwoLevelCacheHierarchy. Use
    the
103             # same clock as the CPUs.
104             system.l2 = l2_cache_class(clk_domain=system.cpu_clk_domain
    ,
105                                        size=options.l2_size,
106                                        assoc=options.l2_assoc)
107
108             system.l2.writeback_clean = True
109             system.tol2bus = L2XBar(clk_domain = system.cpu_clk_domain)
110             system.l2.cpu_side = system.tol2bus.master
111             system.l2.mem_side = system.tol3bus.slave
112     elif options.l2cache:
113 #     if options.l2cache:
114 # 21-Mar-2018 - AKM - Added L3 Cache
```

2.3.1.2.3 Challenge 3: Gem5 Processor Support

The gem5 simulator supports an out-of-order (OoO) processor; however, its implementation is based on five-stage pipeline Alpha 21264 processor [60] with fea-

tures shown in table 2.4. In contrast, processors used in today's computer systems have pipelines with varying stages with advanced features such as vector processors. It is an unimaginable task to make the gem5 OoO processor support all the features of several processors of today, which is a considerable challenge. Therefore, we have constructed all the gem5 simulation-based systems using the only available OoO processor, which is a limitation in our approach.

Table 2.4: Gem5 Simulator O3CPU Model Features

Feature	Value
Pipeline stages	5 Fetch,Decode,Rename, Issue/Execute/Writeback, Commit
Branch Predictor	Tournament (used)
Number of reorder buffer	1
Number of reorder buffer entries	192
Number of load queue entries	32
Number of store queue entries	32
Number of physical integer registers	256
Number of physical float registers	256
Functional Units	IntALU-6,IntMultDiv-3 FPALU-4,FPMultDiv-2 RdWrPort - 4

2.3.2 Selection of Physical Systems

To test the approach on physical systems, we have used physical computer systems with features listed in table 2.5. We have selected these ten physical systems with a range of values for each feature, such as cores, processor clock speed, and in particular, eight systems with x86 instruction-set and two having ARM-based instruction-set.

2.3.3 Applications for Workloads

The categorization of applications according to their compute and data access patterns are shown in [61] [62] categorizing them in compute-bound, memory-bound

Table 2.5: Physical Computer Systems used for Matrix Multiplication, Monte Carlo and Quicksort

Sr	ISA	CPU Speed GHz	Cores	Mem Type	Mem Access MHz	Mem Size GB	L1-L3 Cache Size
1	x86	1.6	2	DDR2	667	2	32-512kB
2	x86	3.2	4	DDR3	1600	4	32kB-6MB
3	ARM	1.2	4	LPDDR3	533	1	8-512kB
4	ARM	1.7	2	LPDDR3	533	2	4kB-2MB
5	x86	3.2	4	DDR3	1600	4	32kB-6MB
6	x86	2.4	12	DDR4	1866	16	32kB-15MB
7	x86	3.2	4	DDR4	2133	4	32kB-6MB
8	x86	3.2	4	DDR3	1600	4	32kB-6MB
9	x86	3.4	4	DDR3	1600	4	32kB-6MB
10	x86	3	2	DDR3	1600	4	32kB-4MB

Configuration taken from following models
1. Intel Core 2 Duo. 2. Intel Core i54460. 3. Qualcomm ARM Cortex A53. 4. Qualcomm snapdragon 600. 5. Intel Core i56500. 6. Intel Xeon E52620. 7. Intel Core i56500. 8. Intel Core i53470. 9. Intel Core i53470. 10. Intel Core i76500U

or compute-plus-memory-bound. We have selected three applications having different compute and data access patterns. A memory-bound application matrix multiplication from linear algebra, a compute-bound application to calculate the value of PI using monte carlo and quicksort, a compute-plus-memory-bound application. Each of these applications was written in C language using known implementation or taken from standard benchmark such as MiBench [63]. Executable binaries of all applications were generated on the same host machine using the GCC compiler for x86 and GCC cross compiler for ARM to eliminate the effect of software environments such as operating systems, optimization variation, etc.

For each application, several problem sizes were considered for simulated and physical systems. For simulated systems dataset, we have used matrix sizes of 50-200 in increments of 50 for matrix multiplication, loop iterations of 100000, 500000, 1000000 to calculate PI using monte carlo, sorting of 1000-6000 words in an increment of 1000 for quicksort. Similarly, for the physical systems dataset, we have used matrix sizes of 300-500 in increments of 50 for matrix multiplication, loop iterations of 100000, 500000, 1000000,5000000 to calculate PI using monte

carlo, sorting of 20000-40000 words in an increment of 5000 for quicksort. We chose smaller problem sizes for the simulated systems dataset than the physical systems dataset because even with these problem sizes, the gem5 simulator took seven days, six days and one and half days continuous runs for matrix multiplication, monte carlo and quicksort applications.

2.4 Feature Encoding and Normalization

The accuracy of a learning-based model primarily depends on the data used in training the model. Machine-learning algorithms such as linear regression require real-valued features with normalized values for higher prediction accuracy, which is the focus of this section.

2.4.1 Encoding Categorical Features

As shown in section 2.3, we have selected a total of nine hardware model features. There are seven real-valued features; processor clock speed, number of processors, L1/L2/L3 cache sizes, memory clock, memory size, and the remaining two categorical features, ISA and memory type having text data. ISA has two possible values but is not limited to; x86 or ARM, while memory type has several possible values such as DDR3, DDR4, LPDDR3, etc. To convert categorical features, ISA and memory type, into real-valued (continuous) features, we applied the dummy encoding scheme as shown.

$$\bar{Y_i} = \alpha + \beta_1 X_{1i} + \beta_2 X_{2i} + ... + \beta_k X_{ki} \tag{2.4.1}$$

$\bar{Y_i}$ = mean of dependent categorical variable for i^{th} group
X_{ji} = value of j^{th} dummy code for i^{th} group
α = mean of referenced group
β_j = mean of j^{th} dummy coded group
When ISA groups values are plugged in, we get

$$ISA = \alpha ISA\_ - 1 + \beta_1 ISA\_0 + \beta_2 ISA\_1 \tag{2.4.2}$$

$$ISA\_x86 = \alpha ISA\_ - 1 + \beta_1(1) + \beta_2(0) = \alpha + \beta_1 \tag{2.4.3}$$

$$ISA\_ARM = \alpha ISA\_ - 1 + \beta_1(0) + \beta_2(1) = \alpha + \beta_2 \tag{2.4.4}$$

Dummy encoding for ISA with two values, x86 and ARM, will have two groups represented by ISA_0, ISA_1 respectively and any other value (base or undefined) as ISA_-1. According to [64], in dummy encoding for regression, intercept (α) will represent the mean of the base value group ISA_-1 and the slops (β_1, β_2) will represent the difference between the means of value groups ISA_0 or ISA_1 and base group ISA_-1. In the simulated systems dataset, there is no base (Undefined) architecture; therefore, none of the values of ISA_-1 will have one resulting in zero means (α). On the other hand, out of 475 simulated systems, 355 are for x86, and 120 are for ARM ISAs resulting in mean values of 0.7473 and 0.2527 for dummy coded groups ISA_x86 (β_1) and ISA_ARM(β_2).

2.4.2 Normalizing Feature Set

After the categorical features were encoded, all the features in our dataset were real-valued. Due to the difference in the range of values of some of the features, mean and variance vary, resulting in poor prediction accuracy. For example, processor clock speed measured in GHz has much lower values (1 to 4.7) compared to L2 cache size in kilobytes (256 to 2048). In this case, the learning model may assign a higher (or lower) weight to L2 cache size than the processor clock speed. A machine-learning algorithm such as a decision tree can perform equally well with a dataset without normalization, but we normalize the dataset to consider it for both linear regression and decision tree. We normalized all the features to bring each feature's values within the range [-1, 1] by applying equations 2.4.5

$$features_{new} = \{f \in features \mid \frac{f - \mu_f}{\sigma_f}\} \tag{2.4.5a}$$

$$features_{new} = \{f_{new} \in features_{new} \mid \frac{f_{new}}{max(abs(f_{new}))}\} \tag{2.4.5b}$$

2.5 Training and Prediction Model

We have used two machine-learning regression models for training and prediction, linear regression [7] [56] and decision tree regression [57] from scikit-learn, a python-based machine learning library [65]. Each of these models are explained in this section.

2.5.1 Linear Regression Model

In the linear regression model, hypothesis is that the dependent variable y (target) is a linear combination of independent variables from input feature set $x_i|1 \leq i \leq n$ as shown in equation 2.5.1

$$\hat{y} = Xw = w_0 + w_1x_1 + w_2x_2 + ... + w_nx_n \tag{2.5.1}$$

In the training phase, random value is assigned for the weight of each feature initially for the samples selected from the dataset for training (train dataset). The weights are then adjusted in each subsequent iterations to predict the target value $\hat{y}$ such that the sum of all errors (ordinary least square) between actual target values y and predicted target values $\hat{y}$ reduces for all samples. The learning model stops iterating when an error is irreducible, which means weight values have been stabilized to have minimum error value, as shown in equation 2.5.2

$$\min_w ||Xw - y|| \quad or \quad \min_w ||\hat{y}(w, x) - y|| \tag{2.5.2}$$

In the prediction phase, only the input features from dataset samples selected for prediction (test dataset) are used in conjunction with learned weights from the training phase to predict the target y value.

2.5.2 Decision Tree Regression Model

In the training phase, the decision tree regression model recursively partitions feature set $x_i|1 \leq i \leq n$ such that the samples with close target values y are on the

same side of the tree. Let dataset at node m be represented by $Q = (x_m, y_m)$ with several possible partitions. For each candidate partition $\theta = (x_{m_i}, t_{m_{x_i}})$, dataset Q is partitioned into $Q_{left}(\theta)$ and $Q_{right}(\theta)$ (equation 2.5.3), where feature x_{m_i} values are compared against the threshold $t_{m_{x_i}}$.

$$Q_{left}(\theta) = (x_m, y_m)|x_{m_i} \leq t_{m_{x_i}} \tag{2.5.3a}$$

$$Q_{right}(\theta) = Q \setminus Q_{left}(\theta) \tag{2.5.3b}$$

For each candidate partition, mean squared error (MSE) $MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \bar{y_i})^2$ is calculated for the left and right partitions which is then used to calculate the impurity function 2.5.4.

$$G(Q, \theta) = \frac{n_{left}}{N_m} MSE(Q_{left}(\theta)) + \frac{n_{right}}{N_m} MSE(Q_{right}(\theta)) \tag{2.5.4}$$

Out of all the candidate partitions, select the one which minimizes the impurity

$$\theta^* = argmin_{\theta(x_m, t_m)}\, G(Q, \theta(x_i, t_i)) \tag{2.5.5}$$

Recursively partition the subsets $Q_{left}(\theta^\star)$ and $Q_{right}(\theta^\star)$ until MSE cannot be improved or maximum allowable depth of the tree is reached $N_m < min_{samples}$ or sample size has reached to one $N_m = 1$. At the end of the training phase, a rule-based tree structure is built where the non-leaf nodes are decision points that guide the direction in which the data path is traversing, for a set of specific feature values of a sample until a leaf node is reached, which provides the target value y for that sample. It is possible to extract these rules from the trained decision tree. The prediction phase uses the tree built in the training phase to predict the target values for the test feature set.

2.6 Experimental Results and Analysis

In this section, we provide the experimental details and analysis of the results. In the first two subsections, we provide performance prediction results from sim-

ulated systems and physical systems. In the third subsection, we evaluate two learning-based models built from linear regression and decision tree machine-learning algorithms for different train and test split ratios for splitting the performance dataset. To establish a generalization of our learning-based model, we applied the model to well-known benchmark programs and provided the result in the fourth subsection.

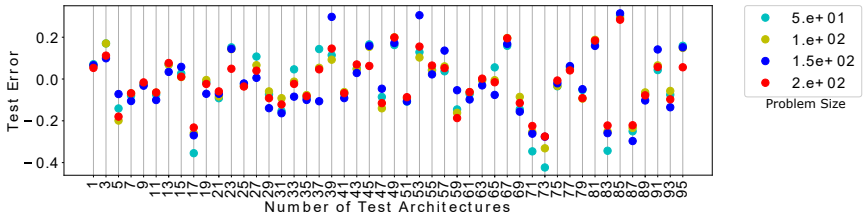
2.6.1 Prediction Accuracy for Simulated Computer Systems

To demonstrate the accuracy of our approach, we have first performed a prediction on hardware models built in the gem5 simulator. We collected runtime from the execution of each of the three selected applications with different problem sizes on all of the 475 simulated systems. For each simulated system, values from nine features were already known because they were used to build the gem5 simulated systems. The performance dataset for each application was built using the systems' hardware features and respective runtimes. We performed encoding and normalization on systems features as described in section 2.4. In the training phase, we have selected 80% of 475 performance dataset samples consisting of normalized system features along with runtime at random (using ShuffleSplit from scikit-learn) to train linear regression and decision tree models, which we call learned-model. From the remaining 20% of performance dataset samples, only the normalized system features were provided to test the accuracy of the learned-model.

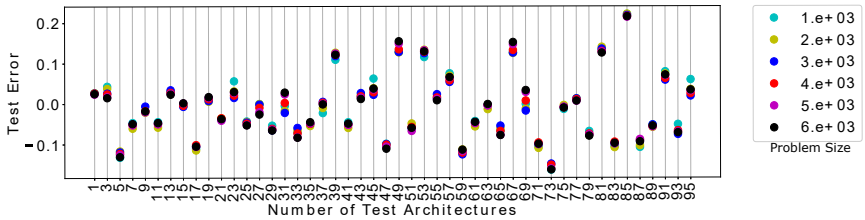
We calculated the test errors as shown in equation 2.6.1 for the test systems by comparing the predicted runtime and actual runtime. Figures 2.3 and 2.4 shows test errors for linear regression and decision tree models respectively.

$$testerror = \frac{y - \hat{y}}{y} \tag{2.6.1}$$

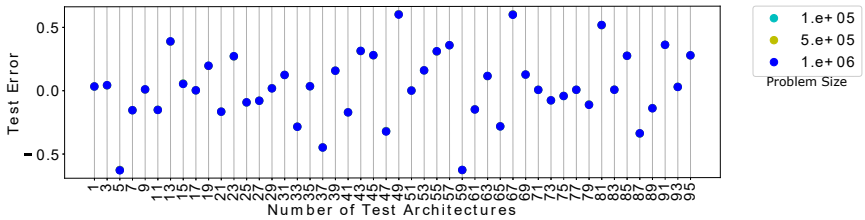
Pearson correlation (r) calculated as per equation 2.6.2 shows the correlation between predicted runtime ($\hat{y}$) values with respect to the actual runtime y values also shown in figures 2.3 and 2.4. Pearson correlation coefficient of more than 90%



(a) matrix multiplication



(b) quicksort



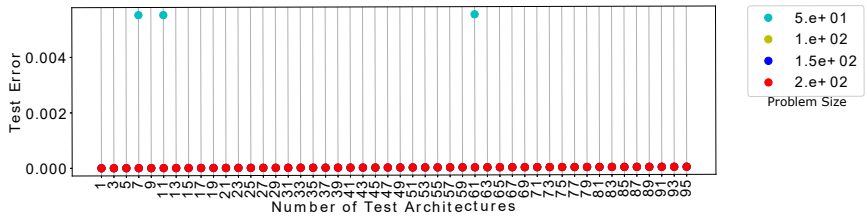
(c) monte carlo

Figure 2.3: Linear Regression (LR) Prediction Error for Simulated Systems Test Architectures per Problem Size (a) Matrix Multiplication [Pearson Coeff: 90.28% to 92.83%], (b) Quicksort[Pearson Coeff: 93.63% to 94.17%], (c) Monte Carlo PI Calculation [Pearson Coeff: 94.87%],

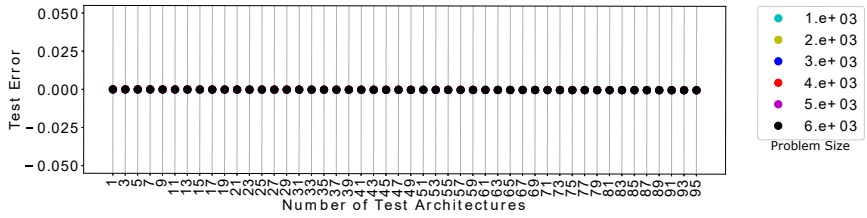
is achieved in case of linear regression whereas the decision tree model achieved 100% confirming higher test errors in case of linear regression compared to decision tree.

$$r = \frac{n(\sum y\hat{y}) - (\sum y)(\sum \hat{y})}{\sqrt{[n\sum y^2 - (\sum y)^2][n\sum \hat{y}^2 - (\sum \hat{y})^2]}} \tag{2.6.2}$$

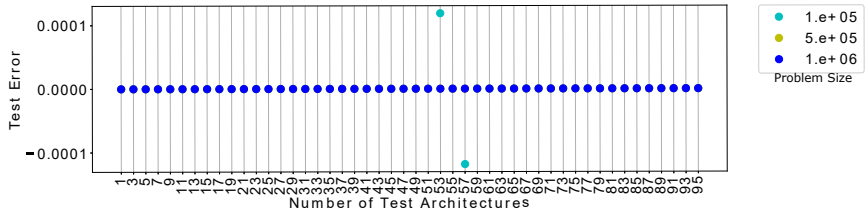
In the case of linear regression, monte carlo has the highest percentage error of -0.6 to 0.6, due to a compute-bound monte carlo depends primarily upon processor features causing it to have much higher variations in runtime. Quicksort application reads a large number of strings to perform sorting depends more on memory features resulting in the smallest percentage error between -0.15 to 0.20. Matrix multiplication, a memory-bound application, depends more on memory features than the processor features having less variation in runtime, resulting in lower error than monte carlo.



(a) matrix multiplication



(b) quicksort



(c) monte carlo

Figure 2.4: Decision Tree Regression (DTR) Prediction Error for Simulated Systems Test Architectures per Problem Size (a) Matrix Multiplication [Pearson Coeff: 100%], (b) Quicksort [Pearson Coeff: 100%], (c) Monte Carlo PI Calculation [Pearson Coeff: 100%]

2.6.2 Prediction Accuracy for Physical Computer Systems

To apply the learning-based model for physical systems, we have used a small set of ten physical systems with diverse system features as depicted in table 2.5. We collected nine system features using dmidecode and lscpu utilities on Unix-based physical systems. We executed the same three applications with multithreaded versions on all ten physical systems using threads one to eight to take advantage of the multicore systems and to increase the dataset size.

To train the regression models, we first performed data encoding for textual features such as mem-type and ISA; then, we normalized the feature set to bring the values for each feature between [-1, 1]. In physical systems, some features do not have any value; hence zero value was used to represent Not a Number (NaN) in a normalized format. For example, physical systems with ARM-based

systems do not have level three cache; hence, the level three cache features were considered all zeros for these physical systems.

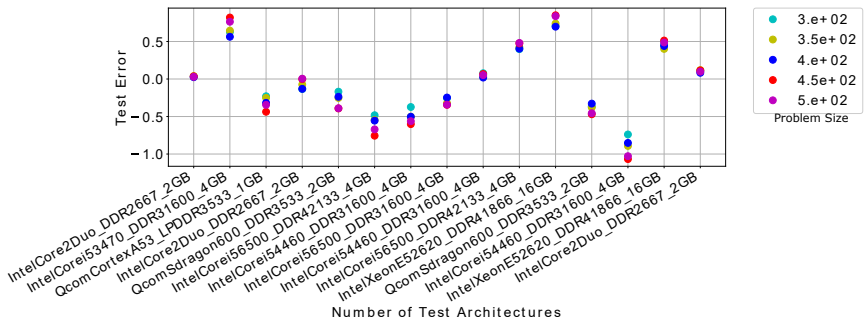
Similar to a simulation-based performance prediction approach, for physical systems also, we have used a train-test split ratio of 80%-20% training and testing (prediction) phase. For the decision tree in figure 2.6, the test errors are showing a similar pattern as observed in the simulated systems for all three applications. Quicksort has the smallest percentage error with most points within 5% that is because of lower runtime variation due to large string data read activity. Matrix multiplication, a memory-bound application, having most of the test errors within 10%, whereas monte carlo, a compute-bound application, having the largest error within the 20% range with few exceptions.

2.6.3 Model Evaluation using Physical Systems Results

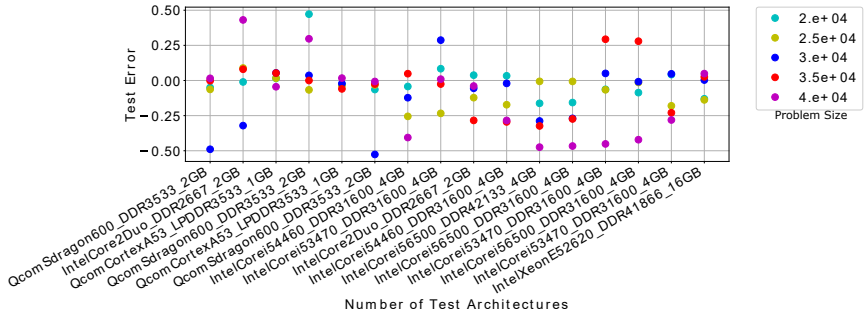
In this section, we have performed a model evaluation for three scenarios. First, we have evaluated the linear regression model compared to the decision tree model and see which one fits the best for the problem at hand. For the second evaluation, we have considered a different ratio to split the dataset in training and testing sets and evaluate the results from each split to identify the split with minimum errors. Finally, we have shown improvements in our performance prediction model by adding an additional system feature.

2.6.3.1 Linear Regression vs Decision Tree

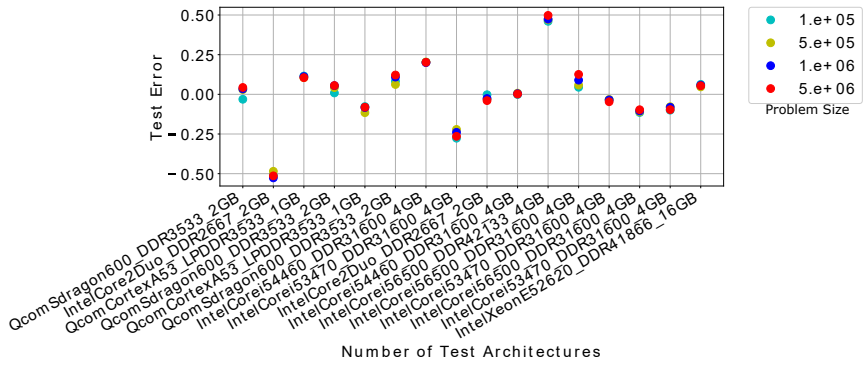
In this section, we have established which of the two regression models, linear regression or decision tree regression, performs better. Figure 2.7 shows the median absolute percentage errors (MedAPE) for each of the three applications from both linear regression and decision tree regression models. We chose the median value instead of the mean value to reduce the effect of outliers as suggested by [66]. We first plotted the histograms using the percentage errors of test systems for all the three applications from linear regression and decision tree models and observe that the percentage errors are not normally distributed. Therefore, we perform a nonparametric Kruskal-Wallis one-way ANOVA test to compare the medians



(a) matrix multiplication



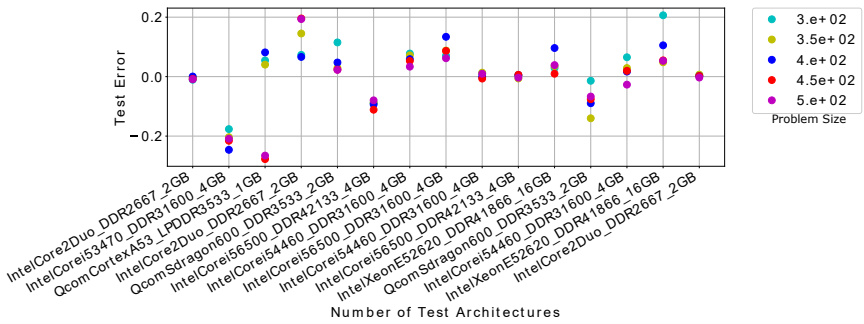
(b) quicksort



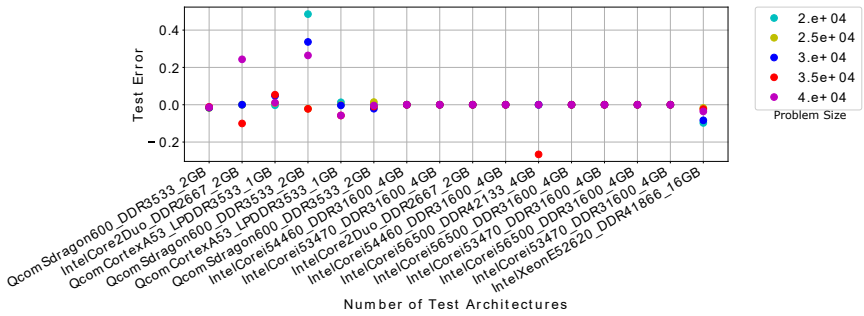
(c) monte carlo

Figure 2.5: Linear Regression (LR) Prediction Error for Physical Systems Test Architectures per Problem Size (a) Matrix Multiplication [Pearson Coeff: 94.84% to 96.17%], (b) Quicksort [Pearson Coeff: 91.67% to 99.88%], (c) Monte Carlo PI Calculation [Pearson Coeff: 98.82% to 98.85%]

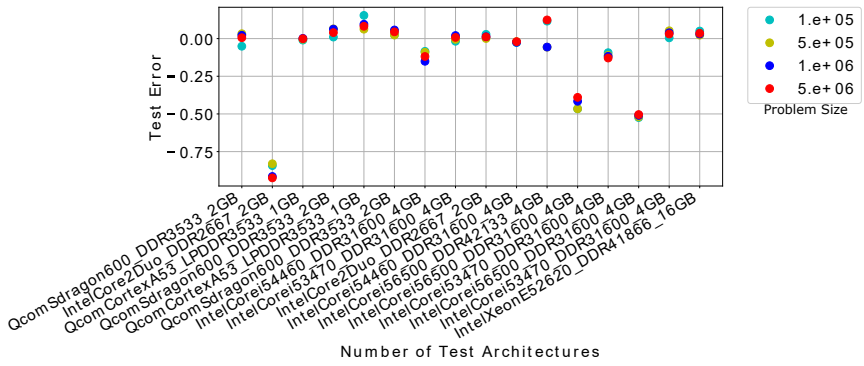
of different problem sizes for each application. In the case of linear regression, we get the p-values of 0.807, 0.553, 0.948 for matrix multiplication, quicksort and monte carlo. While for the decision tree, we get p-values of 0.997, 0.931 and 0.970 for matrix multiplication, quicksort and monte carlo. Furthermore, we observe that the MedAPE for test systems is 39.97%,10.60% and 8.32% for linear regression, as compare to 5.26%, ≈0%, 5% for decision tree model for each of the matrix multiplication, quicksort, and monte carlo applications respectively. Hence, con-



(a) matrix multiplication



(b) quicksort

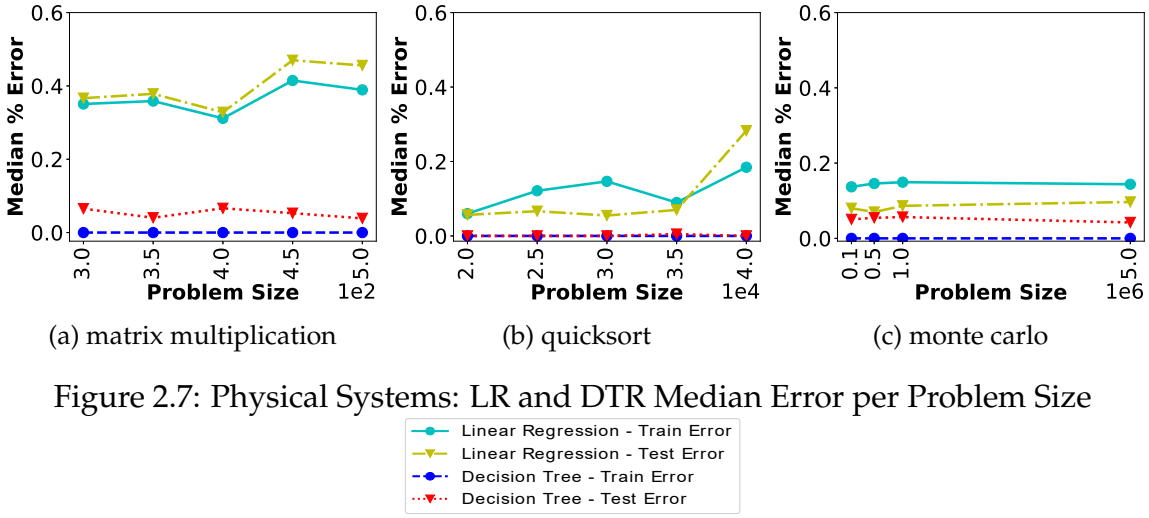


(c) monte carlo

Figure 2.6: Decision Tree Regression (DTR) Prediction Error for Physical Systems Test Architectures per Problem Size (a) Matrix Multiplication [Pearson Coeff: 97.83% to 99.40%], (b) Quicksort [Pearson Coeff: 91.82% to 99.98%], (c) Monte Carlo PI Calculation [Pearson Coeff: 98.52% to 98.87%]

clusively the decision tree model has much higher accuracy and is a better fit than linear regression for the problem at hand with a given dataset.

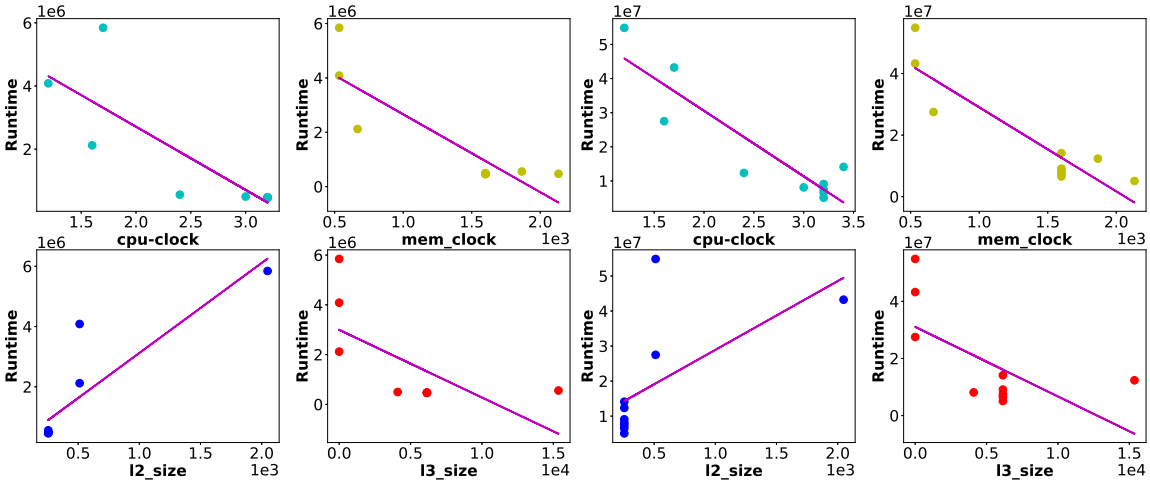
To understand the reason behind this, we have plotted four of the system features CPU clock, memory clock, L2 cache size and L3 cache size on the x-axis and the actual runtime on the y-axis for matrix multiplication and monte carlo as shown in figure 2.8. It can be inferred from this plot that the runtime is not linearly dependent on the input features. In other words, we cannot find weight W



(a) matrix multiplication (b) quicksort (c) monte carlo

Figure 2.7: Physical Systems: LR and DTR Median Error per Problem Size

in $y = Wx$ to make runtime linearly dependent on feature x. The quicksort plot is not shown, but it shows a similar non-linear relationship between hardware features and runtime.



(a) matrix multiplication (b) monte carlo

Figure 2.8: Input Features vs Runtime

It is observed that matrix multiplication has a higher error in linear regression than the other applications. To further analyze this point, we considered four input features, namely cpu clock, memory clock, l2 size and l3 size, as shown in figure 2.8. We trained four different linear regression models with each of the four features and the actual runtime for each application. The plots show the original runtime on Y-axis and the hardware feature on X-axis for two applications, matrix

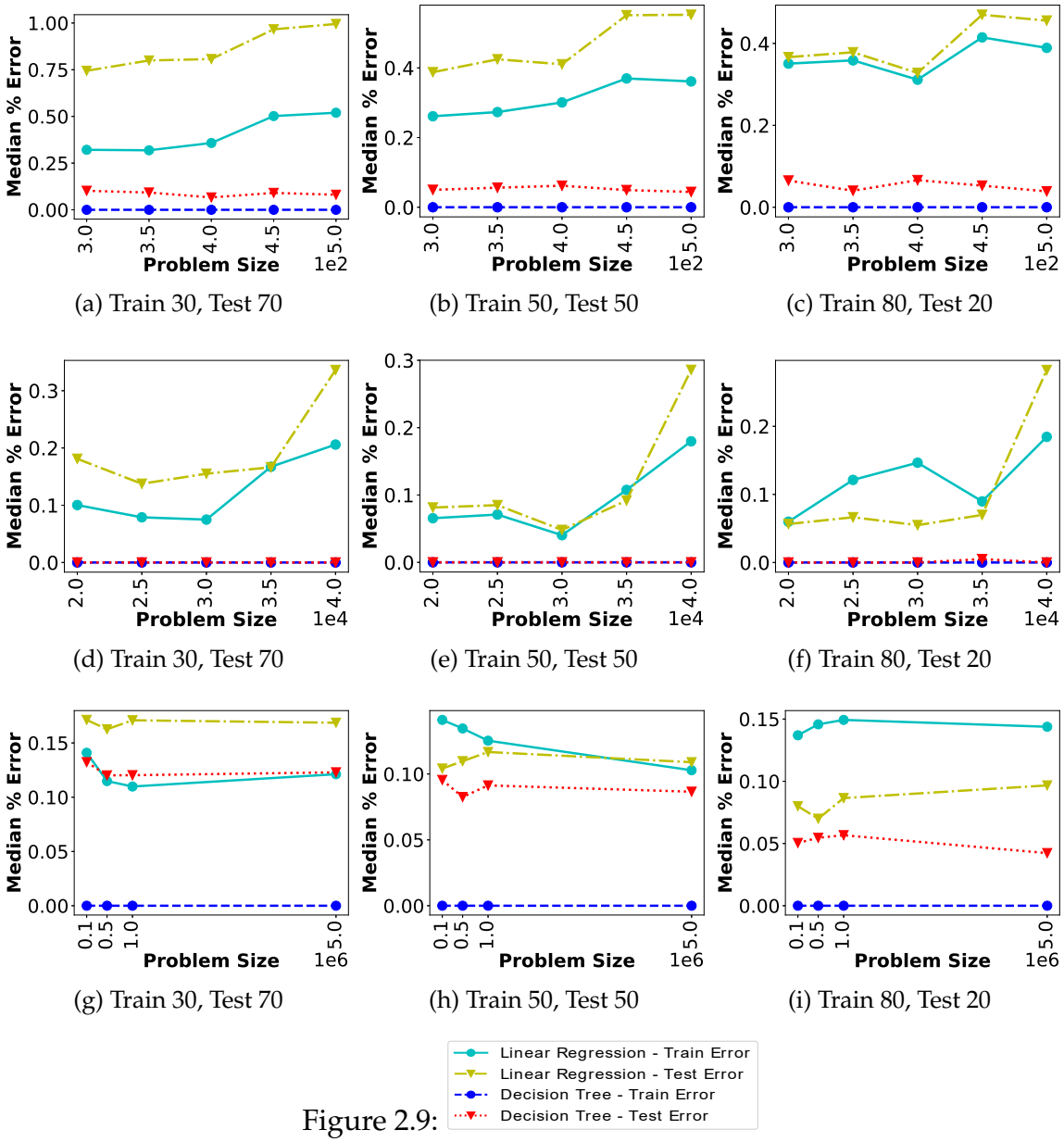
multiplication and monte carlo. The line in the plots indicates the predictions from the respective linear model. It is clear from the plot that the distance from the actual runtime to the predicted value, which is the projection on the line, is higher in matrix multiplication than monte carlo. This is especially more evident in the plots with hardware features cpu clock and memory clock. The higher the distance, the larger the error; therefore, matrix multiplication has a higher error than monte carlo. These results are indicative of the higher error in matrix multiplication from the multiple linear regression model used.

2.6.3.2 Train vs Test Dataset Split Ratio

To test the prediction accuracy of the learning-based model, we need to carefully split the dataset into train and test sets. During the training phase, a learning-based model can learn with a higher prediction accuracy but can have poor prediction accuracy for testing dataset, a problem known as over-fitting [67]. Over-fitting is the problem when the learning-based model works too hard to find accurate relationships between independent input features and dependent output during the training phase, but it fails to establish a generalized relationship. Hence, when the prediction is performed for the testing dataset, the model may have higher errors as the test input features may not exactly coincide with the training dataset features. To decide the train-test split ratio, we explored several options and have plotted three of them in figure 2.9.

According to median absolute percentage errors (MedAPE) for testing dataset in decision tree regression; matrix multiplication errors were 7.2% for 30-70 split, 5.2% for 50-50 split and 5.2% for 80-20 split. Similarly, for monte carlo errors were 12.2% for 30-70 split, 8.9% for 50-50 split and 5% for 80-20 split. For quicksort, MedAPE is $\approx$0 because the MiBench quicksort benchmark program that we have used reads a large number of strings to sort from a file resulting in much less variance in runtime making it more deterministic for prediction. Our learning-based model has the highest accuracy with the 80-20 train-test split, and hence we have used the same for final models.

We have two observations from these results. First, the monte carlo application



(a) Train 30, Test 70 (b) Train 50, Test 50 (c) Train 80, Test 20

(d) Train 30, Test 70 (e) Train 50, Test 50 (f) Train 80, Test 20

(g) Train 30, Test 70 (h) Train 50, Test 50 (i) Train 80, Test 20

Figure 2.9: Median absolute percentage error for different train-test split (a)-(c) Matrix Multiplication (d)-(f)Quicksort (g)-(i) Monte Carlo

has larger errors as compared to matrix multiplication and quicksort. We believe this is due to the manufacturer's variability in the processors. For memory-bound applications such as matrix multiplication, the impact on the runtime of manufacturer variability of a processor is hidden due to slower memory access, but for monte carlo, due to a higher number of computations, the effect of manufacturer variable is seen persistently. The second observation is that the four Intel core i5 systems with similar system feature values have different runtime values re-

sulting in higher errors in some cases. This indicated that possibly one or more system features might not have been considered in the original dataset that distinctly identifies all the four Intel Core i5 models.

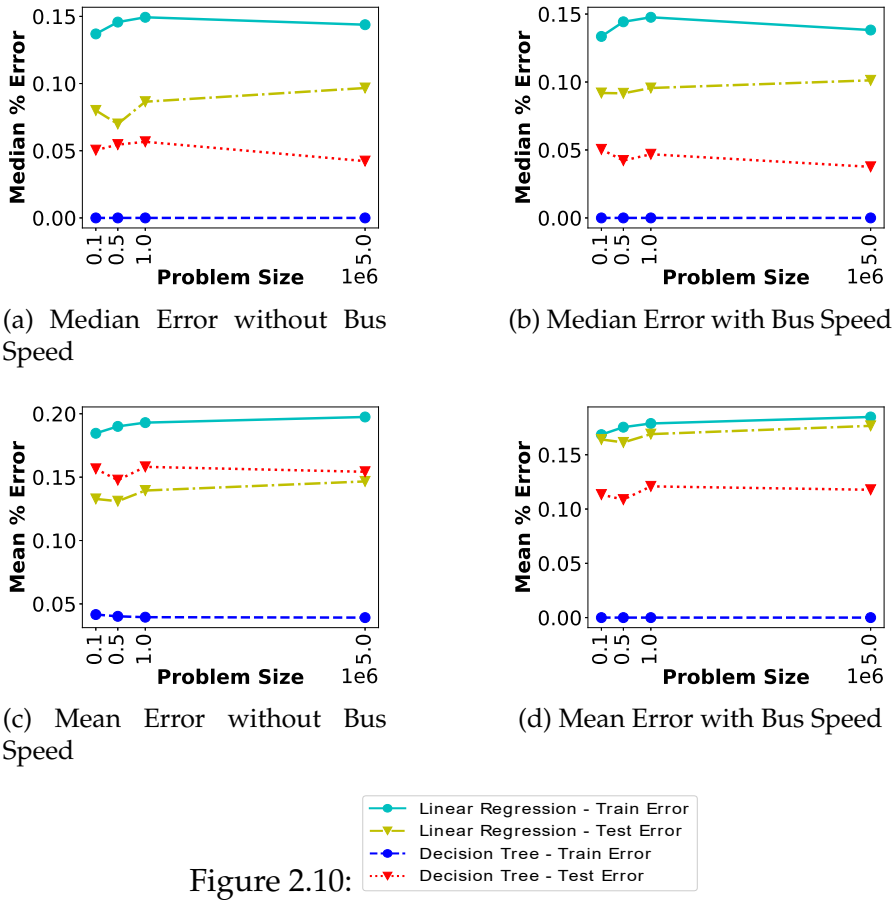
2.6.3.3 Additional Architecture Features for Prediction Improvement

To resolve the issue of higher percentage errors in monte carlo in the second observation, we reviewed the processor documentation of Intel Core i5 models from [68] to identify additional hardware feature(s) that can be used to distinguish Intel Core i5 models. After careful consideration, we added "bus speed" as an additional feature to be considered for the learning-based model. We calculated the bus speed in megabytes per second (MB/s) using the information provided at [68] and [69] for each of the Intel processors. For ARM-based systems, bus speed was considered as the default value of zero.

Figure 2.10 shows monte carlo's median (b) and mean (d) percentage errors with bus speed and median (a) and mean (c) percentage errors without the inclusion of bus speed. After the addition of bus speed, the median errors improved from 5% to 4.4% for monte carlo that is a 12% improvement in prediction. Mean test errors for monte carlo improved even higher at 23%, with 15.42% without the bus speed and 11.77% with the bus speed. The reduction in errors with the addition of bus speed is due to the fact that the learning-based model is able to differentiate between the different Intel Core i5 processors. The addition of bus speed also improved the prediction accuracy of matrix multiplication and quicksort in the range of 15% to 30%.

2.6.4 Performance Prediction of Mantevo mini-applications and NAS Parallel Benchmarks

To assess the true effectiveness and generality of our proposed approach, we applied a learning-based performance prediction model on two NAS Parallel Benchmarks (NPB) benchmark programs, EP and MG, and the miniFE benchmark program from the Mantevo suite. According to NPB documentation, EP is embarrass-



(a) Median Error without Bus Speed

(b) Median Error with Bus Speed

(c) Mean Error without Bus Speed

(d) Mean Error with Bus Speed

Figure 2.10: Comparison of Median and Mean absolute percentage error for Monte Carlo with and w/o Bus Speed

ingly parallel (compute-bound), and MG is memory-bound applications. These benchmark programs are representative of real-world applications with multiple and complex phases. We have selected problem sizes of 25, 50, 75 and 100 for the miniFE benchmark and problem sizes of S, A and B for EP and MG benchmarks for performance prediction. As far as system selection is concerned, we selected two server-like systems with Intel Xeon processors with many cores and large memory, three Intel Core i7 systems, three Intel Core i5 systems and two ARM-architecture based embedded boards to represent commonly used computer systems. Detail configurations of these systems are listed in table 2.6.

We executed the MPI versions of each of these three benchmark programs with the respective problem sizes on all ten physical systems. To take advantage of hyper-threading, we executed the number of processes for each application from one to two times the number of cores in each system; that is, we executed 24 appli-

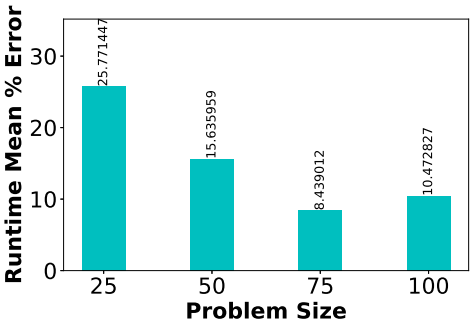
Table 2.6: Physical Computer Systems used for EP, MG and miniFE

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	32kB-6MB
2	ARM	1.2	4	LPDDR3	533	1	8kB-128kB
3	ARM	1.7	2	LPDDR3	533	2	4kB-512kB
4	x86	3	2	DDR3	1600	4	32kB-4MB
5	x86	2.6	2	DDR3	1066	4	32kB-4MB
6	x86	3.2	4	DDR3	1600	4	32kB-6MB
7	x86	3.2	4	DDR4	2400	4	32kB-6MB
8	x86	3.2	4	DDR4	2666	16	32kB-12MB
9	x86	2.4	12	DDR4	2133	64	32kB-15MB
10	x86	2	16	DDR3	1600	32	32kB-20MB

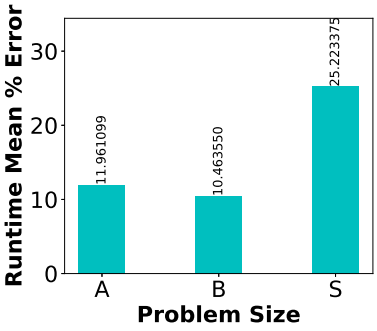
Configuration taken from following models
1. Intel Core i56500. 2. Qualcomm ARM Cortex A53 3. Qualcomm snapdragon 600 4. Intel Core i76500U 5. Intel Core i7620M 6. Intel Core i53470 7. Intel Core i56500 8. Intel Core i78700 9. Intel Xeon E52620v3 10. Intel Xeon E52640v2

cation processes on a system with 12 cores. We collected system features using the dmidecode utility and extracted runtime for each execution from the benchmark logs. We performed training and prediction for linear regression and decision tree models using an 80-20% train-test split ratio after feature normalization.

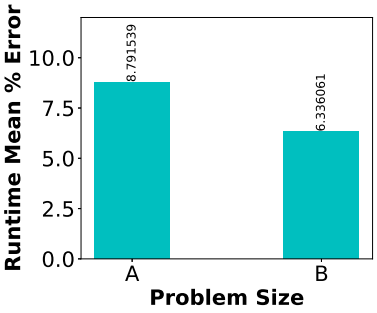
The performance prediction results for benchmark programs also confirmed that decision tree regression performed better than linear regression in line with our earlier experiments. Figure 2.11 shows the decision tree model prediction error of about 8-15% for EP and miniFE and about 6-8% for MG. For smaller problem sizes of 25 in miniFE and S for EP, percentage errors are higher because runtimes for small problem sizes have a higher variance. In the case of MG, with problem size S, runtime values were zero for many systems; hence the bar is not shown in the chart. We observe that EP being compute-bound similar to monte carlo has a higher prediction error due to manufacturer variability, whereas MG being memory-bound similar to matrix multiplication, has a lower prediction error.



(a) miniFE Prediction Error



(b) EP Prediction Error



(c) MG Prediction Error

Figure 2.11: miniFE, EP and MG Prediction Errors Per Problem Size

2.7 Threats to Validity and Limitations

- Internal Validity: To ensure internal validity, we executed different types of applications on a large number of simulated systems and available physical systems. To build 475 simulated systems, we spent a couple of months collecting the hardware features from real systems, making required modifications to the gem5 source code, and then building simulated systems using these collected hardware features. We carefully selected ten systems from many available systems for constructing physical systems datasets, each with disparate hardware feature values. Furthermore, to reduce the effect of non-determinism in physical systems, we executed each application 25 times to collect runtime values and have used the mean runtime value as the actual runtime (performance). We collected runtime for each thread for parallel applications and summed it up for calculating the final runtime. We have also evaluated the machine learning model using different train-test split ratios to ensure that the higher prediction accuracy ratio is used in the

final model.

- External Validity: To ensure external validity, we utilized the same host computer to compile all applications and generate binaries (executables), which we executed on all systems. Since the host machine used for compilation has an x86 instruction-set, we used a cross compiler to compile applications for systems based on the ARM instruction set. Additionally, we selected physical systems with the Unix-based operating system (mostly Ubuntu) to reduce the impact on measured performance due to the dissimilarity in operating systems.

- Limitations: Our learning-based model relies on several assumptions. First, we assume that the behavior of actual performance is steady. However, due to variations in the system's load, performance measurement may vary vastly, making learning difficult from this unsteady performance. Furthermore, our assumption is that the performance of simulated systems is proportional to that of physical systems with identical hardware features, which they are up to a certain extent. But because of the design differences in the simulator, such as the support of only one out-of-order processor with 5 stage pipeline, the system represented by the simulated system may not precisely match the real system.

2.8 Summary

This chapter proposed a learning-based technique to select a hardware system best suited for a given software by predicting the performance of the software on unseen computer systems without running the same software. This was achieved by learning relationships between systems hardware features and the actual runtime of a given application. We identified system features from a processor, cache, and memory hierarchy that dictated the performance of compute-bound, memory-bound, or compute-plus-memory-bound applications. We demonstrated that using the training dataset, the learning-based model can capture the relationships between computer systems' hardware features for a given application and actual

runtimes. The learned model then predicts the performance of the same software for unseen systems.

The performance prediction result can be used for the selection of the computer system by forming a performance-based cluster of the systems while considering the performance of a given software on all the computer systems. A single cluster will have systems with similar performance for a given software. Applications that are embarrassingly parallel, such as EP from the NPB benchmark, will have better performance on a server-like system with many-core even with lower clock speed, whereas highly sequential applications will have better performance on systems having fewer cores with higher clock speed. Memory-bound applications such as MG from the NPB benchmark will have better performance on systems with higher memory bandwidth and low latency.

Our proposed learning-based model is general in nature. One can extend this model for performance prediction of multi-node systems by including network features. To use our model from figure 2.1 for multi-node systems, each system shown will become multi-node homogeneous clustered systems with each node having the same processor, cache and memory features connected via a network. The system features will include network features such as bandwidth in addition to the processor, cache, and memory features. Benchmarks such as the NAS Parallel Benchmark can be executed to collect the runtime on a homogeneous cluster. Performance prediction of the same benchmark programs can be performed for multi-node clusters with different node and network features.

Below are conclusions from our work in this chapter:

1. We demonstrated that a single performance prediction model can predict the performance of systems of different instruction-set such as ARM and x86-based systems with different features of various hardware components such as processors, cache and memory.

2. Our prediction model worked on both simulated systems and for the physical systems. On simulated systems, the prediction accuracy of 99% was achieved, whereas, on the physical systems, the prediction accuracy of more than 90% was achieved considering 80% systems for training and 20% for

testing.

3. Prediction accuracy in simulated systems was much higher compared to physical systems. This is because runtimes from the simulation-based approach are deterministic due to the lack of interference in shared resources. While the runtimes from physical systems are non-deterministic due to interference from multiple tasks running at the same time.

4. Our performance model predicted the performance of compute-bound, memory-bound, and compute-plus-memory-bound applications with an accuracy of more than 90%.

5. The proposed prediction model considered system features with continuous values (real-valued) and categorical values. Categorical values were non-numeric, which were converted to numerical features with the help of the encoding technique.

6. Two Machine learning regression models, linear regression, and decision tree were applied for performance prediction. We observed that due to the non-linear relationship between hardware features and the runtimes, the decision tree regression model gave at least 40% higher accuracy than linear regression for all the three applications matrix multiplication, quicksort, and monte carlo.

7. We had an option of choosing from several machine learning algorithms. Machine learning algorithms such as linear regression and decision tree have useful analytical and computational properties. These properties help to understand the decision-making process for prediction and make them computationally less expensive but with small accuracy compromise. Complex models such as neural networks provide higher accuracy but are generally difficult to understand the prediction outcome and are computationally more expensive. Therefore, we first started with a linear regression model, but prediction accuracy was poor; hence, we used the decision tree model, which provided us the results with acceptable accuracy.

8. Dataset was split with different ratios in training and testing set (70-30, 50-50, and 80-20). We observed that an 80-20 split gave a prediction error of 5%, which was much lower than the prediction error from the other splits.

By providing accurate performance prediction, work provides insight into the selection of computer systems that fit the performance criteria of a user. For the selection of computer systems, three factors are important; performance, power, and hardware cost. In this work, we have primarily focused on the performance aspect. In future work, we plan to include other factors such as power and hardware cost. We would also like to explore additional machine learning models such as a neural network in subsequent work.

CHAPTER 3

Evaluating Machine Learning Models for Disparate Computer Systems Performance Prediction

3.1 Overview

Performance modeling is an active area for research to estimate the performance of a software application for computer systems with the known processor, cache and memory features. The estimated performance (runtime) from the performance model aids us with the tool to select system(s) with optimal performance. Thus, performance modeling is useful to software developers and system architects to choose the most optimal application implementation and features of the underlying computer systems. Performance modeling is primarily implemented using supervised machine learning algorithms. The accuracy of estimated (predicted) performance depends on the machine learning algorithm utilized for modeling and the nature of the performance dataset.

Several research works have focused on performance prediction models utilizing various machine learning algorithms. The study in [5] evaluates linear, nearest neighbor, support vector, gaussian, tree-based and neural network-based machine learning algorithms for performance models of scientific applications with skewed and irregular domain configurations on four leadership class HPC platforms. Work in [21] deals with predicting optimal cloud configurations for HPC applications before deployment using linear, tree-based and neural network-

based machine learning algorithms. The prediction of power and performance on heterogeneous systems using neural networks is studied in [18]. The prediction accuracy rate of 97.5% is achieved in [17] using multiple neural networks and PCA on SPEC CPU2006 and SPEC CPU2017 benchmarks. The work in [14] utilizes linear and tree-based machine learning algorithms to predict the performance on multicore cloud systems with shared resource contention.

We identified the following questions from related work: (i) The referenced works use 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 simulated systems as compared to physical systems?. We further analyzed different configurations of neural network models to answer the following questions: (i) Does the data scaling affect the performance prediction accuracy of neural network models? (ii) How does the accuracy of one-layer and multi-layer neural network models differ? (iii) For benchmark programs with different computations and memory access patterns, will neural networks require different configurations? In this chapter, we address these questions to understand the performance model better using different machine learning models.

The remainder of the chapter is organized as follows: Section 3.2 describes the computer hardware selection and dataset. Section 3.3 describes machine learning models that we have used for the study. Section 3.4 provides three scenarios: (a) Overall performance of models on each dataset and discussion on the inferences that can be drawn. (b) Evaluation of results concerning compute-bound and memory-bound applications. (c) Analysis of performance concerning physical and simulated systems for each model. Section 3.5 evaluates neural network models with different configurations. Finally, section 3.6 provides concluding remarks and work that we plan to continue.

3.2 Dataset Preparation

3.2.1 Applications Selection for Workload

We used two categories of applications, compute-bound and memory-bound, to test the applicability and accuracy of the performance prediction models. The selected four compute-bound applications are monte carlo application to calculate the value of PI, maximally stable regions (mser) application from San Diego Vision Benchmark Suite (SD-VBS) [70], miniFE [59] from the Mantevo mini-applications benchmark suite, and embarrassingly parallel (EP) application from the NAS parallel benchmark (NPB) [58]. Similarly, three selected memory-bound applications are matrix multiplication, feature tracking (tracking) application from SD-VBS [70], and multi-grid on a sequence of meshes, long and short distance communication (MG) application from the NPB [58].

3.2.2 Computer Systems Selection

To execute the selected applications, we selected the computer systems that describe the class of computers commonly used today. We chose 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 3.1. For the simulated systems dataset, we utilized the gem5 simulator, a widely accepted simulator for architectural research. We built 120 simulated systems based on the ARM instruction set (ISA) and 355 simulated systems based on the x86 ISA using the system-call emulation mode of gem5 as described in chapter 2.3.1. We considered nine system features for the construction of gem5 simulated systems, as shown in table 2.1. These feature values were gathered from the real memory and processor models commonly used in today's computers.

We executed applications listed in the "Scientific Applications" column on 475 simulated systems and 8 physical systems as listed in the "System Type" column in table 3.2. For physical systems, each application was executed for the number of processes from one to two times the number of cores in the system; that is, we

Table 3.1: Physical Computer Systems used for Evaluation of Machine Learning 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
Configurations were taken from the 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

have executed 24 application processes on a system with 12 cores. This was to take advantage of systems with hyper-threading. We extracted runtimes for each execution from the benchmark logs.

Table 3.2: Applications used as workloads for Evaluation of Machine Learning Models

Applications	Benchmark	System Type	(MB/CB)	Data Points
matrix multiplication		Physical	MB	280
matrix multiplication		Simulated	MB	1900
monte carlo		Simulated	CB	1425
monte carlo		Physical	CB	224
mser	SD-VBS	Simulated	CB	475
tracking	SD-VBS	Simulated	MB	475
miniFE	Mantevo	Physical	CB	124
EP	NPB	Physical	CB	108
MG	NPB	Physical	MB	108

CB = Compute-Bound (compute-intensive)
MB = Memory-Bound (data-intensive)

3.3 Machine Learning Models

Performance modeling based on empirical methods is implemented using supervised machine learning algorithms. A labeled dataset is used to make a machine learning algorithm learn the function $\Im x$ which maps inputs X_i to an output y_i. Here X_i is a multidimensional vector or tuple while the output y_i is a single numerical value. In our case, each sample input X_i is a multidimensional tuple with hardware features corresponding to processor and memory as shown in table 3.1 for physical systems and table 2.1 for simulated systems and the output y_i is runtime or performance of the application. Different machine learning algorithms derive the said function $\Im x$ with dissimilar prediction accuracy, and the one which gives the most accurate predicted y values is used for modeling.

The overview of the models with various machine learning algorithms that we use in this chapter is as follows.

3.3.1 Support Vector Regressor (svr)

Support vector regression [71] 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 kernel space to make better predictions. The svr uses the $\epsilon - insensitive$ loss function and penalizes predictions that are far from the actual output.

3.3.2 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: $y = \alpha + x\beta + \epsilon$ whereas a multi-variable or multiple linear regression model takes the form: $y = \alpha + x_1\beta_1 + x_2\beta_2 + ... + x_k\beta_k + \epsilon$,where y is a dependent variable, x is a single input feature value in the simple regression model, and $x_1, x_2, \ldots, x_k$ are the multiple input feature values in the multiple regression model.

3.3.3 Ridge Regression (rr)

Ridge regression [72] is an extension for linear regression when a penalty term is added to the loss function to avoid over-fitting or when there is multicollinearity among the input features. Loss function $L = \sigma(\hat{y}_i - y_i)^2 + \lambda(\sigma\beta^2)$ is used, where λ is a penalty term for regularization and β are the coefficients to be estimated.

3.3.4 K Nearest Neighbors (knn)

K nearest neighbors regression [73] belongs to the class of instance-based methods, where the model is not built explicitly during the training phase but determined only during the prediction of value. KNN uses the concept of neighbor instances which are determined based on the distance from X for which the prediction is required. It selects the user-defined number of neighbors and averages their actual y values to give us the predicted y value for a given X.

3.3.5 Gaussian Process Regressor (gpr)

Gaussian process regression [74] model implements the gaussian process for regression purposes. This algorithm finds the normal distribution, which maximizes the log marginal likelihood using an optimizer parameter. The gaussian process gives the predictive variance estimate around the prediction and a clear probabilistic interpretation.

3.3.6 Decision Trees Regressor (dt)

Decision tree regression [57] belongs to the class of recursive partitioning methods. The decision tree is used to build regression models that are similar to a binary tree data structure. At each non-leaf node of the tree, permutations of feature values and their respective thresholds are compared. Each feature-threshold combination is one possible data split into left and right sub-trees. For each data split (feature-threshold combination), the impurity function is calculated using the mean squared errors between the actual output values and the possible pre-

dicted values of data from the left and right sub-trees. A combination of feature and threshold with the lowest mean squared error (MSE) is chosen to split the data at each non-leaf node. With the final construction of the tree, leaf nodes provide the predicted values. The depth of the final tree can be limited by an argument called max_depth.

3.3.7 Random Forest Regressor (rf)

Random forest [75] 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 set of associated rules for dividing the dataset into smaller subsets. It outputs the mean prediction of the individual trees that it builds. This technique of using multiple models to obtain higher predictive performance is called the ensemble model.

3.3.8 Extremely Randomized Trees (etr)

Extremely randomized trees [76], or extra trees is like a random forest. But, it does not bootstrap data items, meaning it samples the data without replacement. Furthermore, node splits are random 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 the random forest.

3.3.9 Gradient Boosting Regressor (gbr)

Gradient boosting regression [77] is a boosting technique where weak learners are combined to get a strong learner. We have used decision trees as weak learners. The 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 tree's parameters and moving in the direction to minimize the error.

3.3.10 XGBoost (xgb)

XGBoost [78] stands for eXtreme gradient boosting. It is an optimized variant of the gbr model that practices parallel programming, pruning 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 a sparse aware model.

3.3.11 Deep Neural Networks

A deep neural network (DNN) is an artificial neural network with multiple hidden layers which estimate the linear or non-linear relationship between input and output. We used four DNN variants as shown in table 3.3. Neurons column indicates the number of neurons used at each layer and params column shows the total number of parameters which is derived by $Neurons_{currLayer} * (Neurons_{prevLayer} + 1)$. 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 introducing non-linearity, whereas the linear activation function is used for summing up the predicted performance value from the last hidden layer to the output layer. The number of epochs used is 100 for each variant.

Table 3.3: Model Summary of DNN Variants

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

3.4 Experimental Evaluation/Results

In this section, we performed experiments to evaluate machine learning models described in section 3.3 using gem5 simulated systems and physical systems performance datasets from section 3.2. The performance datasets consist of two categorical hardware features, isa and mem-type, with text data that is not acceptable by many machine learning algorithms. Hence, we converted these categorical features data from text to real values using one-hot encoding. Furthermore, we normalized the dataset using $StandardScaler()$ function from scikit-learn, a python-based machine learning library [65]. To ensure that the model provides an accurate prediction for random selection of samples from the performance dataset for training and testing, we have used ten-fold cross-validation using the $ShuffleSplit()$ function from the scikit-learn library.

We trained each machine-learning model with 80% of the dataset and the remaining 20% is used for prediction due to higher accuracy for 80:20 ratio. For example, for tree-based models dt, rf, etr and gbr, matrix multiplication and NPB EP on physical systems have a mean error of 4.1% and 12.54% for 60:40 whereas 3.08% and 10.33% for 80:20 ratio respectively. We have used R^2 score, median absolute percentage error (MedAPE), and mean squared error (MSE) metric to measure the accuracy of models. We took the median over the mean absolute percentage error values that we got from cross-validation for calculating the MedAPE score. Low values of MedAPE and MSE represent that the model has high prediction accuracy, whereas the R^2 score of 1 indicates the model fits the data well. The negative R^2 score indicates a worse model fit than a horizontal hyperplane.

We evaluated machine learning models in three different scenarios. First, the overall result for each model averaged over all nine datasets. Second, a comparison of model performance concerning the types of applications and finally, a comparison of model performance between physical and simulated systems performance datasets.

3.4.1 Model comparison for performance prediction

Figure 3.1 shows the mean value of all datasets' R^2 scores and the MedAPE score. We observed that the order of model performances according to the prediction accuracy 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 got after comparing all three plots in figure 3.1 and after taking a majority vote of all three metrics, namely R^2, MedAPE, and MSE scores. The explanation for this behavior in performance concerning each model is explained below.

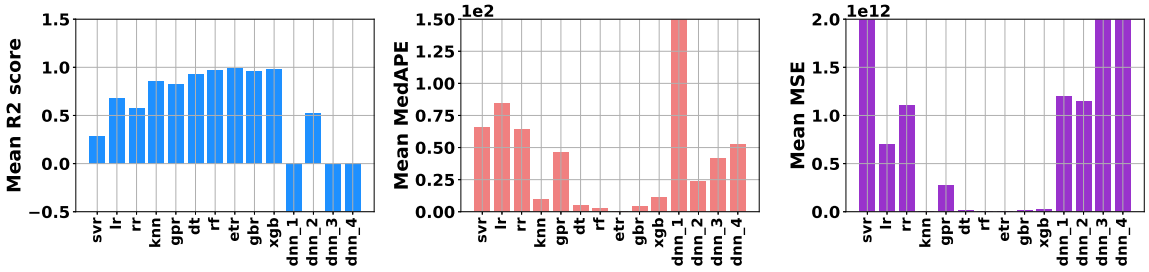


Figure 3.1: Mean of R^2, MedAPE and MSE values per model for all datsets

- ***svr***: Support vector regressor with mean R^2 = 0.28 and mean MedAPE = 65.91% (figure 3.1), 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 the mean R^2 = 0.68 (figure 3.1) has poor accuracy because the runtime is not linearly dependent on all the memory and processor features. The ridge regression regularizes the overfitted linear model that can be observed in figure 3.1 where MedAPE for linear regression is 84.72% whereas for ridge regression is 64.02%.

- ***knn***: KNN regressor with mean R^2 = 0.86 and mean MedAPE = 9.44% (figure 3.1) performs better than linear models and kernel svr. The parameters

n_neighbors and distance metrics play an important role in the prediction accuracy of KNN which we optimized using grid search. The best distance metric for our data is Minkowski, which is a generalization of both euclidean and manhattan distance. The accuracy of KNN indicates that the performance dataset has systems with alike features in high dimensional vector space.

- ***gpr***: Gaussian process regressor with mean R^2 = 0.83 and mean MedAPE = 46.29% (figure 3.1) also performs better than linear models and kernel svr. But still, tree-based models have higher prediction accuracy. The reason for reduced accuracy in gpr 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 does not follow the multivariate gaussian distribution and is not tractable by the model.

- ***dt***: Decision tree regressor with mean R^2 = 0.93 and mean MedAPE = 4.95% (figure 3.1) 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 the dataset features, the decision tree structure provides higher prediction accuracy.

- ***rf and etr***: Random forest (rf) and extra tree regressor (etr) tree-based models provide the highest prediction accuracy. Both of these algorithms are bagging approaches of the decision tree, in which a large number of trees improve the accuracy of the model. The etr with mean R^2 = 0.99 and mean MedAPE = 0.69% (figure 3.1) performs better than rf with mean R^2 = 0.97 and mean MedAPE = 2.58%. This is because etr model uses a random data split that provides diversification in binary-tree generation during training strengthening the etr model.

- ***gbr and xgb***: Boosting algorithms, gbr and xgb, are based on weak learners

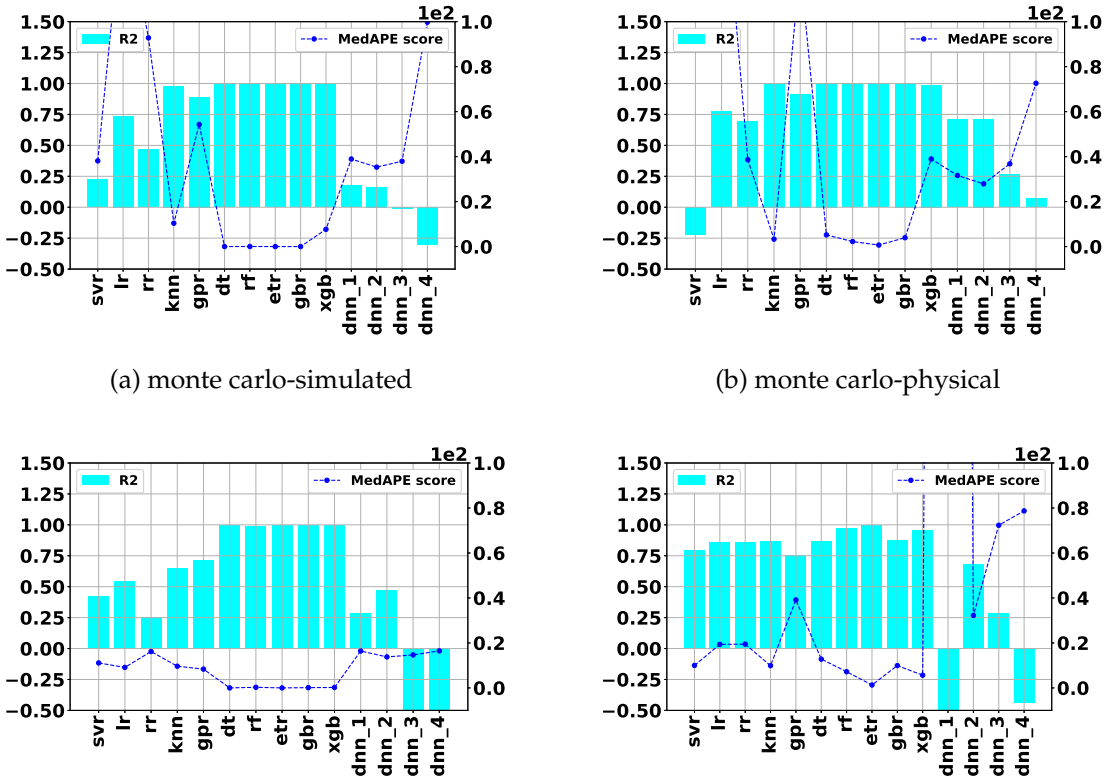
(high bias and low variance). A gbr with mean R^2 = 0.96 and mean MedAPE = 4.02% (figure 3.1) performs better than xgb with mean R^2 = 0.97 and mean MedAPE = 11.57%. These boosting models reduce the bias to get good accuracy. But the base learner decision tree has good prediction accuracy with low bias. 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 dnn_2, a best performing model among the four variants (table 3.3), with mean R^2 = 0.53 and mean MedAPE = 23.98% (figure 3.1) has lower prediction accuracy. We believe that this is due to a limited number of data points available, as shown in table 3.2), neural network models have lower accuracy than best performing tree-based models.

3.4.2 Estimating the Effect of Performance concerning Compute-Bound and Memory-Bound Applications

Figures 3.2 and 3.3 shows R^2 and MedAPE scores on left and right y-axis with ranges $[-0.5, 1.5]$ and $[-10\%, 100\%]$ respectively for compute-bound and memory-bound applications datasets (refer table 3.2).

The first observation from figures 3.2 and 3.3 is that five models, 'dt', 'rf', 'etr', 'gbr' and, 'xgb' have mean R^2 score of 0.75 and mean MedAPE score below 20% for nearly all scientific applications. The result demonstrated that these models are a good fit for datasets listed in table 3.2. Therefore, we have considered only these five tree-based models for subsequent prediction accuracy analysis. The second observation from figure 3.4 is that an average taken over MedAPE values of the tree-based models, the prediction accuracy in memory-bound applications (with 3.75% mean error) is slightly better than in compute-bound applications (with 5.58% mean error). This is because the processors have higher manufacturer variability due to complex architecture design than memory.



(a) monte carlo-simulated

(b) monte carlo-physical

(c) mser-simulated

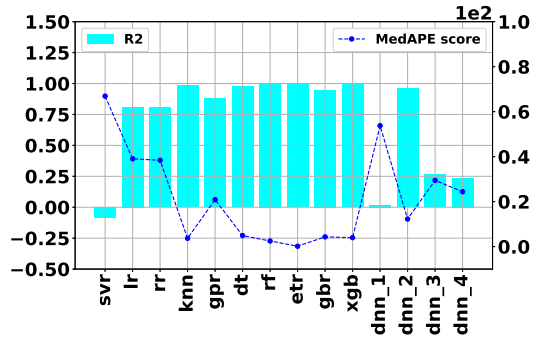
(d) miniFE-physical

Figure 3.2: R^2 score and MedAPE for Compute-Bound Applications

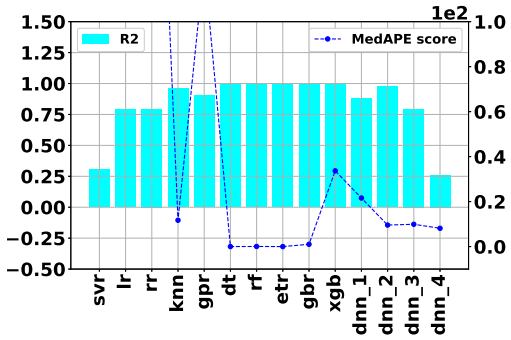
3.4.3 Comparing model performance on Physical and Simulated Systems

Figure 3.5 shows R^2 score and MedAPE gathered from 10-fold cross-validation for performance prediction on physical and simulated systems for matrix multiplication (memory-bound) and monte carlo (compute-bound) applications. The models that do not show box-plot have much higher error falling outside the y-axis range.

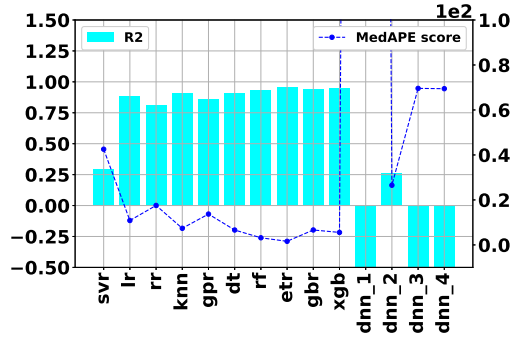
By comparing the performance of tree-based models, it was observed that the prediction accuracy for simulated systems (with mean R^2 = 0.999 and mean MedAPE = 4.20%) is higher than the accuracy for physical systems (with mean R^2 = 0.989 and mean MedAPE = 7.16%). This is due to the larger variance in runtime for non-deterministic physical systems than for deterministic simulated systems. We also found that memory-bound applications such as matrix multiplication



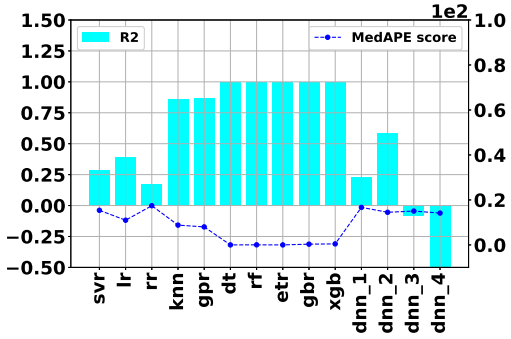
(a) matrix multiplication-physical



(b) matrix multiplication-simulated



(c) MG-physical



(d) tracking-simulated

Figure 3.3: R^2 score and MedAPE for Memory-Bound Applications

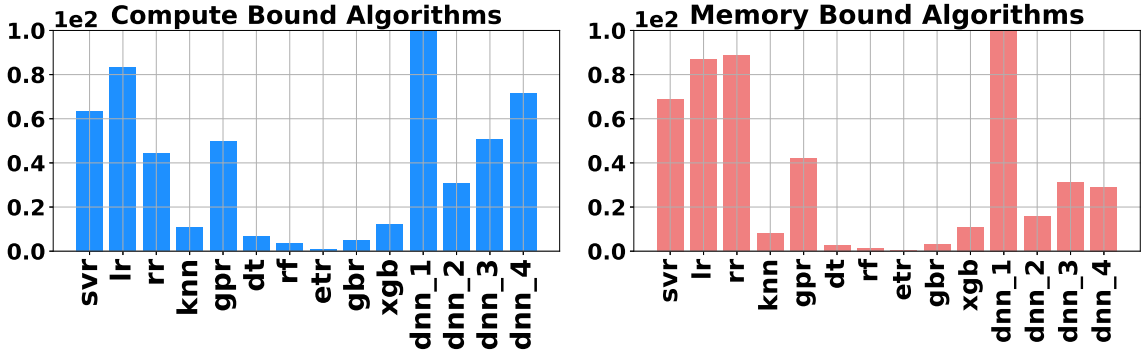
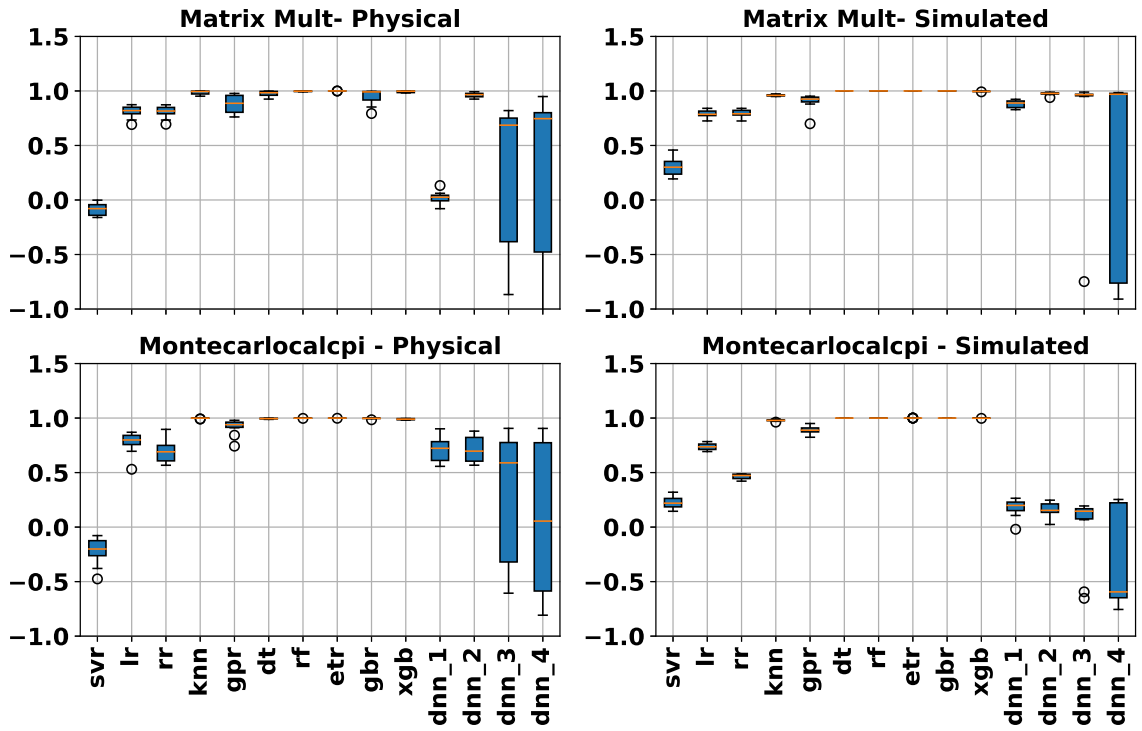
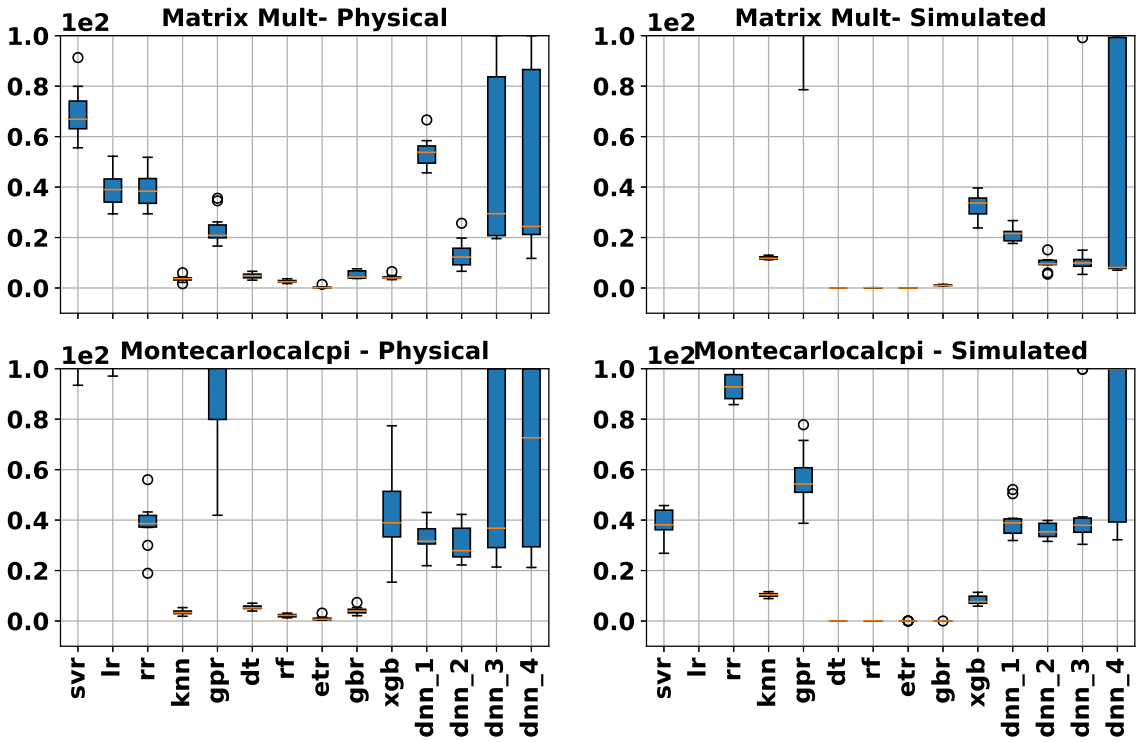


Figure 3.4: Mean MedAPE for both CB and MB Applications

(with mean R^2 = 0.98 and mean MedAPE = 3.46%) has higher prediction accuracy then compute-bound applications such as monte carlo (with mean R^2 = 0.99 and mean MedAPE = 10.85%), when executed on physical systems. This is because the higher variance in runtime contributed by manufacturer variability from the complex architecture of processors in the physical system results in higher error in compute-bound applications. On the other hand, we observed that for simulated



(a) R^2 score



(b) Absolute Percentage Error

Figure 3.5: Comparing model performance on Physical and Simulated Systems

systems, compute-bound applications (monte carlo with mean R^2 = 0.9997 and mean MedAPE = 1.69%) have higher prediction accuracy then memory-bound applications (matrix multiplication with mean R^2 = 0.990 and mean MedAPE = 6.71%). Higher prediction accuracy for compute-bound applications in simulated systems is caused by the availability of a single out-of-order five-stage pipeline processor in gem5 which is used to build all 475 simulated systems.

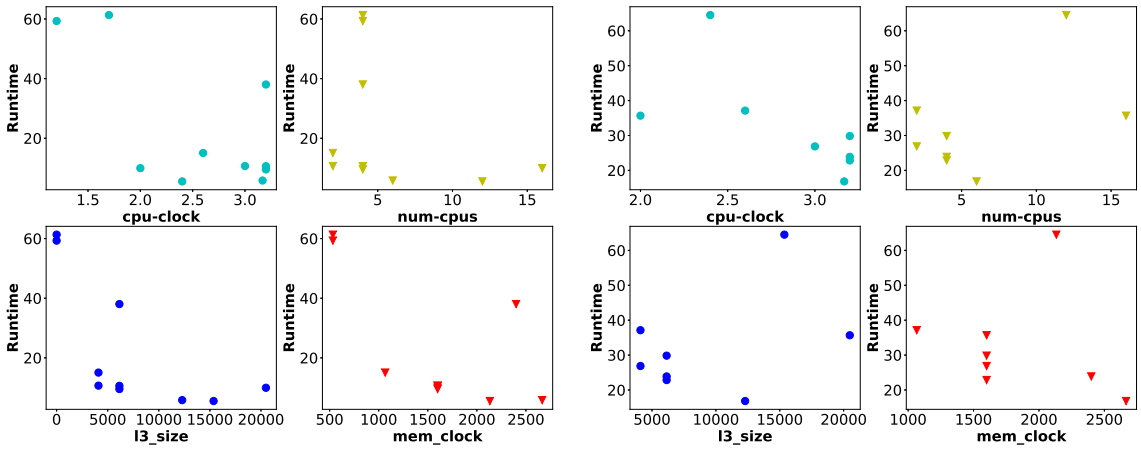
3.5 Evaluation of Neural Network Models

In this section, we evaluate different neural network models.

3.5.1 Neural Network Models

Neural network models are commonly used to learn from a dataset with non-linear relationships between dependent and independent variables. Figure 3.6 shows the non-linear relationship between the performance (runtime) of an application and the system hardware features cpu-clock, num-cpus, l3-size, and mem-clock for the systems on which application is executed. Therefore, we tackled this problem using an artificial neural network (ANN) in this work. We have used a dense fully-connected sequential ANN model from Keras framework [79] consisting of one input layer, one output layer, and one or more hidden layers. We explored two different ANN models, one has only one hidden layer (One-Layer Model) and the other with three hidden layers (Multi-Layer Model), as shown in figure 3.7. Each ANN's input layer has the number of neurons equal to distinct hardware features such as processor clock speed, L1/L2/L3 Cache sizes, memory size, collected during dataset preparation as an input for performance prediction. The output layer has only one neuron since we predict one output, the performance of an application. Each hidden layer of the multi-layer model has half of the neurons from the predecessor hidden layer starting with 400, 200, and 100 neurons in the first, second, and last hidden layers.

Both these neural networks are represented by the following mathematical notations:



(a) miniFE (b) EP

Figure 3.6: Hardware Features vs Runtime

Input Features X = X1, X2, ..., Xm

X1 = Processor Clock Speed

X2 = cores

X3 = L1 Cache Size,...,

Xm = Memory Type

One-Layer NN Predicted Performance having one hidden layer with n nodes:

$\hat{y} = \sum_{j=1}^{n} W'j(\sum_{i=1}^{m} WijXi)$

Multi-Layer NN Predicted Performance having 3 hidden layers with n, n/2 and n/4 hidden layer nodes:

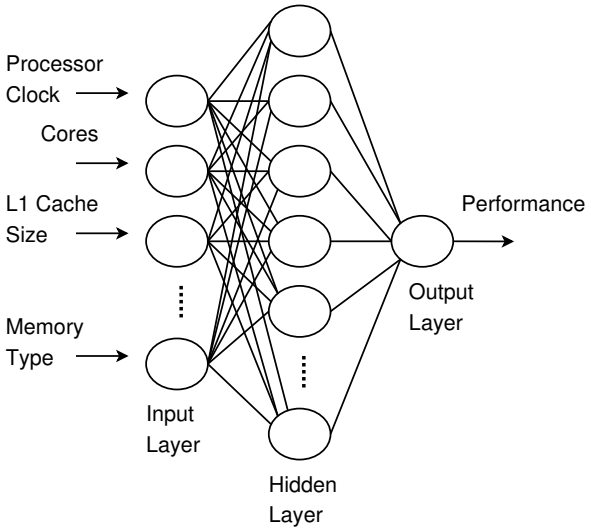
$\hat{y} = \sum_{l=1}^{n/4} W'''l(\sum_{k=1}^{n/2} W''kl(\sum_{j=1}^{n} W'jk(\sum_{i=1}^{m} WijXi)))$

It is important to know the time complexity of each neural network model because a neural network model with lower time complexity having comparable prediction accuracy is a preferable model for performance prediction. For our models, the time complexity is represented by the following notations:

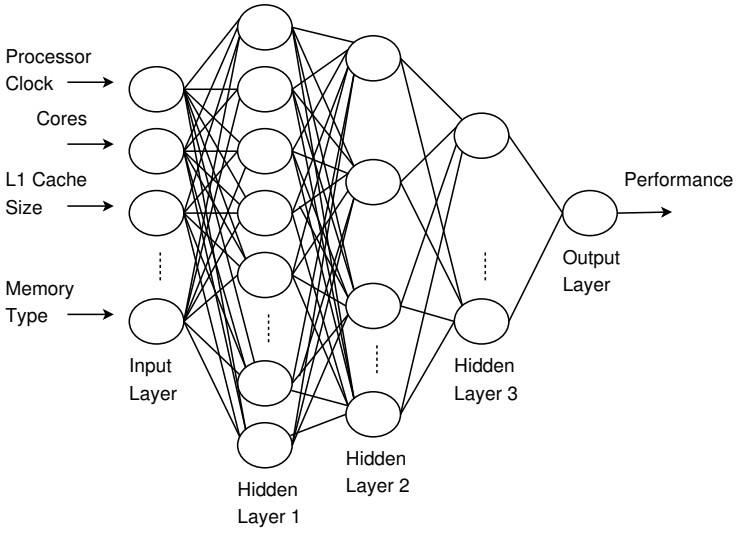
t = number of samples in the dataset

e = number of epochs considering Back Propagation Algorithm

Tmn = Time Complexity for Calculating Weights for Hidden Layer with m nodes



(a) One-Layer NN



(b) Multi-Layer NN

Figure 3.7: Fully-Connected Neural Network Models

to n nodes.

$Tmn = m * n$

One-Layer NN Predicted Performance having a one hidden layer with n nodes: T = (calculate weights for input to hidden layer + calculate weights for hidden layer to output layer) * number of training samples * number of epochs

$T = (m * n + n * 1) * e * t = et * (mn + n)$

Multi-Layer NN Predicted Performance having 3 hidden layers with n, n/2

and n/4 hidden layer nodes:

T = (calculate weights for input to 1st hidden layer + calculate weights for 1st to 2nd hidden layer + calculate weights for 2nd to 3rd hidden layer + calculate weights for 3rd hidden layer to output layer) * number of training samples * number of epochs

$T = (m * n + +n * n/2 + n/2 * n/4 + n/4 * 1) * e * t$

$= et * (mn + n^2/2 + n^2/8 + n/4)$

Performance prediction is a regression problem that estimates the parameters by minimizing the sum of squares taken over all the responses and all the observations. This is a least-squares problem, therefore, we have used mean squared error ($MSE(y, \hat{y}) = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$) loss function for our ANN. An activation function is responsible for transforming summed weighted input from the previous layer to the next layer until the output layer. Rectified Linear Unit (ReLU) is used as an activation function in hidden layers for non-linearity. Simultaneously, the linear activation function is used, to sum up, the predicted performance value from the last hidden layer to the output layer.

3.5.2 Experimental Evaluation of Neural Network Models

We first built the performance dataset $[X, y]$ for each selected benchmark program EP, MG, and miniFE by executing them with different processes on eight systems shown in the table 3.1. We collected the hardware feature values X of nine features from physical systems using dmidecode utility and collecting the performance (runtime) y from the benchmark logs. We encoded two features instruction-set-architecture (ISA) and memory type with text data using one-hot encoding resulting in a real-valued feature set X'. We then scaled the encoded performance dataset $[X', y]$ with an appropriate scaling method. We divided the scaled performance dataset $[X', y]$ into train-test split with 80-20 ratio using train_test_split() function of scikit-learn that shuffles the data using ShuffleSplit() before the split resulting in training set $[X'_{train}, y_{train}]$ and testing set $[X'_{test}, y_{test}]$. Finally, we evaluated the neural network models built using Keras library by training them on the

training set $[X'_{train}, y_{train}]$ and performing prediction on the testing set $[X'_{test}, y_{test}]$.

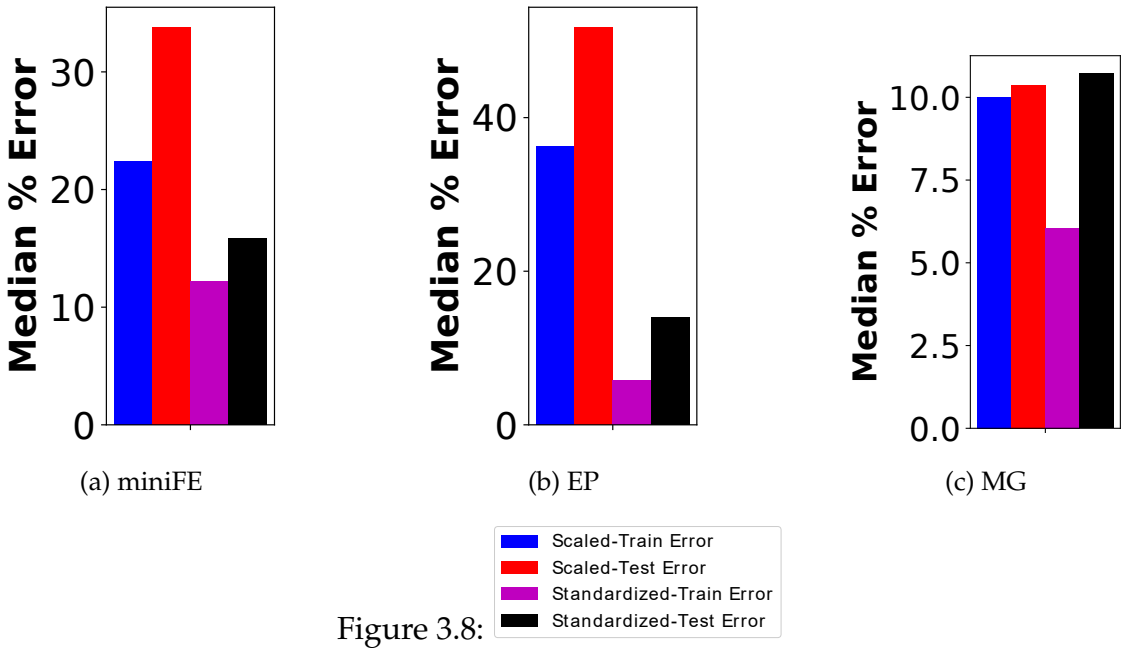
For each of the experiments in this section, we built neural network models five times and have shown percentage errors as quartiles of box plots for accurate evaluation. The training and prediction times for one layer neural network model with 100 epochs and 3000 neurons with 80-20 train-test split are 1 sec and 1 msec. On the other hand, the same data split for a multi-layer neural network model with 100 epochs and 400, 200, and 100 neurons in the first, second and third hidden layers respectively took 3 secs and 1 msec. We performed neural network model experiments on a Lenovo laptop with two cores, two threads per core, Intel Core i7-6500U processor, and eight GB DDR3 memory.

In this section, we performed different experiments and evaluated the neural network models described in section 3.5.1 using datasets from section 3.2. We evaluated the ANN models in three different ways. First, we looked at the impact of data scaling on performance prediction by the neural network model. We then evaluated the two ANN models one-layer versus multi-layer, and finally, we evaluated the model for selecting the number of neurons and optimization function.

3.5.2.1 Impact of Data Scaling

Our performance dataset has two categorical features instruction-set and memory type with text data, which is not accepted by the neural network. We applied one-hot (dummy) encoding [64] using categorical encoding python package to convert categorical features into real-valued features. Machine learning algorithms such as neural networks perform better or converge faster when used with a scaled dataset. we evaluated our ANN model's prediction accuracy for MinMaxScaler and StandardScaler from the scikit-learn framework [65]. MinMaxScaler transforms the dataset such that each feature values are scaled in a range between 0 and 1 by applying $(X - X_{min})/(X_{max} - X_{min})$ to each feature X. Mean and variance for each feature remain unchanged in the case of MinMaxScaler. On the other hand, StandardScaler applies $(X - \mu(X))/\sigma(X)$ to standardize each feature by removing mean and scaling to unit variance. In other words, all features have a mean of zero and variance of one once standardized.

We trained multi-layer neural network models separately for the scaled and standardized dataset. We used 80% of the dataset for training, and the remaining 20% dataset was used as a test set for predicting the performance. Figure 3.8 shows mean percentage errors (MPE) for training and testing phases. We observed that the accuracy of multi-layer ANN with a standardized dataset is at least 50% higher in most cases than scaled data. In the compute-bound EP application, the impact of a standardized dataset is higher than the memory-bound MG application due to manufacturer variability. Processors have higher manufacturer variability due to complex design as compared to simpler memory modules. The compute-bound applications' performance depends more on the processor features, whereas the memory-bound applications' performance depends more on the memory features. Therefore, the manufacturer variability effect in processors is more evident in compute-bound applications.



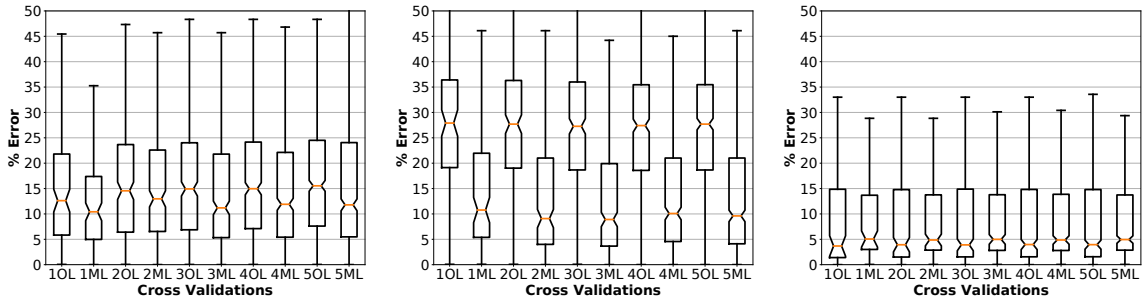
(a) miniFE (b) EP (c) MG

Figure 3.8: Scaling Effect on Performance Prediction on Neural Network Model

3.5.2.2 One-Layer vs Multi-Layer Neural Network

Cross-validation [80] is a common technique to evaluate the accuracy of machine learning models. We have used 5-fold cross-validation with ShuffleSplit, an iterator that splits the data into train and test sets after shuffling the data randomly. We

evaluated both one-layer and multi-layer neural network models as described in section 3.5.1 with an 80-20% train-test ratio for each of the five folds. We modeled a one-layer model with 3000 neurons in the only hidden layer and a multi-layer model with 400-200-100 neurons in each of the three hidden layers. Figure 3.9 shows the percentage error from one-layer and multi-layer for each of the five folds. For the NPB EP, the one-layer model has a median error of about 27-28%, whereas the multi-layer model has 7-12%. Similarly, for Mantevo miniFE, a one-layer model median error is 13-16% compared to 11-13% for multi-layer. For NPB MG, median errors for both one-layer and multi-layer models are between 4 to 5%.



(a) miniFE (b) EP (c) MG

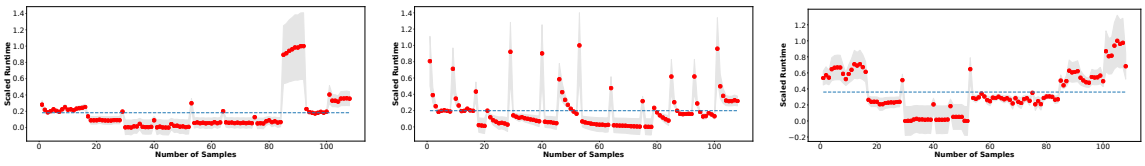
Figure 3.9: Cross-Validation Prediction Error for One-Layer (OL) VS Multi-Layer (ML) Neural Network

We concluded that the multi-layer model has higher accuracy in compute-bound applications and has similar performance for memory-bound applications compared to the one-layer model. The multi-layer model's higher prediction accuracy is because the one-layer model acts like linear regression with one hidden layer having a linear activation function, whereas the problem of performance prediction is non-linear due to the non-linear relationship between hardware features and performance (runtime). The multi-layer neural network model is able to learn the non-linear relation effectively than the one-layer model resulting in lower errors in the multi-layer model.

Due to the lower error in the multi-layer model compared to the one-layer model, we have used a multi-layer model for performance prediction experiments in subsequent sections.

3.5.2.2.1 Number of Neurons

First, we evaluated the multi-layer neural network model to understand the number of neurons required for convergence of predicted performance to the actual performance to reduce the prediction error depending upon the type of application, compute-bound, or memory-bound. Figure 3.10 displays runtime error for each sample as a shaded region along with the mean line for each of the three applications. We observed that NPB EP, a compute-bound application, has higher runtime variations to its mean, resulting in higher errors. For example, the NPB EP has the highest error with 100 neurons compared to miniFE and NPB MG. Therefore, the NPB EP requires a higher number of neurons in the multi-layer neural network to converge, as shown in figure 3.11. On the other hand, the reduction in error is also higher in the case of EP than miniFE as we increased neurons from 100 to 700 in steps of 200. For NPB MG, a memory-bound application, convergence happens much earlier at 50 neurons, and therefore, an increase in the number of neurons further increases the error rather than reducing it. In the case of miniFE, the mean line is close to many samples except with samples from 85 to 90, placing it in-between EP and MG, resulting in a higher error as compared to EP but lower error than MG. Therefore, convergence for miniFE takes a smaller number of neurons than EP but larger than MG.

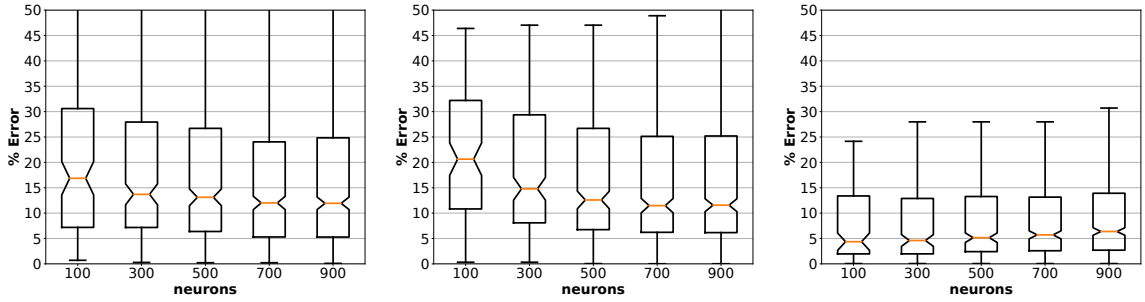


(a) miniFE (b) EP (c) MG

Figure 3.10: Variations in Runtime with Respect to Mean

3.5.2.2.2 Optimizer Selection

The optimizer function minimizes the objective function (error function) in a neural network by parameter adjustment. In other words, the optimizer function adjusts internal parameters (weights and biases) so that the error between the actual value and the predicted value is reduced, minimizing the overall loss dur-



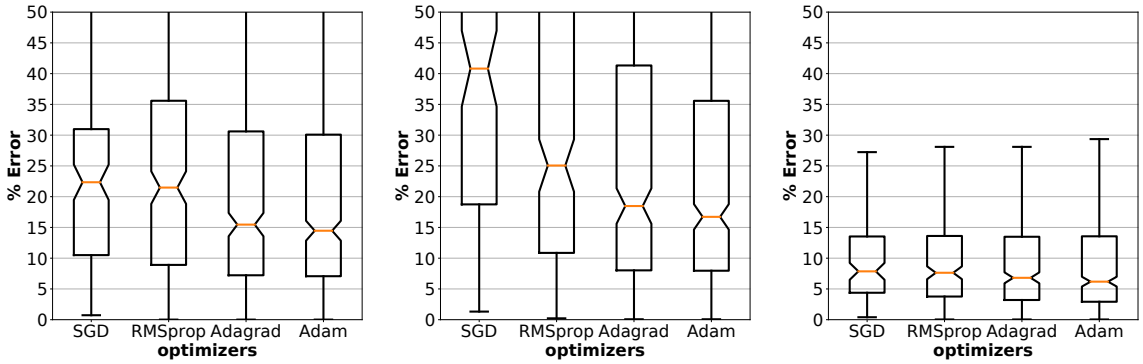
(a) miniFE (b) EP (c) MG

Figure 3.11: Effect of Neurons on Performance Prediction

ing each iteration. We evaluated our multi-layer neural network model with four optimizers from Keras; Stochastic Gradient Descend (SDG), AdaGrad, RMSProp, and Adam as shown in figure 3.12. We observed that Adam has the lowest errors of 13%, 17%, and 6% for miniFE, EP, and MG applications, respectively. Performance prediction can be a non-convex problem where a neural network can find multiple local minima of runtime values for convergence due to the hardware features for test systems can be close to more than one systems' hardware features used during neural network training. SDG oscillates with the same momentum through a gradient to find the optimum value of runtime, resulting in either jumping over the local minima or not being able to converge fast enough resulting in higher errors. RMSProp uses a learning rate that changes according to change in the slope to find the minima changes. The change in learning rate allows RMSProp to adapt to the change in slope for convergence and find local minima faster than SDG. Adagrad further improves convergence with parameter-specific learning rates, which is inversely proportional to parameter updates. Finally, Adam provides parameter-specific learning rate and bias correction according to [81], resulting in the lowest errors from the four optimizers studied in this experiment.

3.6 Summary

The work in this chapter analyses the prediction accuracy of 14 machine learning algorithms and different configurations of neural network models built on the performance dataset from gem5 simulated systems and physical systems for



(a) miniFE (b) EP (c) MG

Figure 3.12: Effect of Optimizers on Performance Prediction

seven applications. The conclusions from the experiments are as follows:

- The tree-based models, including tree-based bagging models dt, etr, and rf outperform all other machine learning models, including the deep learning variants.
- The performance prediction on memory-bound applications have slightly better accuracy than compute-bound applications due to manufacturer variability.
- After exhaustive comparison of prediction accuracy from performance prediction models built from physical systems and simulated systems datasets, it is concluded that models from simulated systems have higher accuracy compared to physical systems due to non-deterministic physical systems.
- System hardware features have a non-linear relationship with performance, hence, the linear models have lower prediction accuracy compared to other models. Furthermore, the random forest model performs better than the decision tree model due to averaging error approach of the bagging method in random forest.
- For neural network models, standardizing features with StandardScaler have lower errors than MinMaxScaler because StandardScaler scales each feature with zero mean and unit variance making them fit the normal distribution

curve for the neural network to apply weight in a similar range for each feature to find global or local minima.

- In evaluating the one-layer versus multi-layer model, we demonstrated that the multi-layer model has better prediction accuracy because the one-layer model with linear activation function acts like linear regression while hardware features against runtime have a non-linear relationship.
- We established that compute-bound applications such as NPB EP take longer to converge due to higher runtime variations compared to memory-bound applications such as MG. Therefore, compute-bound applications require a higher number of neurons.

Our future work includes building efficient multivariate models. We are planning to include power in addition to runtime for multivariate prediction. We aspire to contribute more datasets and also intending to apply transfer learning for cross-platform and cross-systems performance prediction.

CHAPTER 4

Multivariate Performance and Power Prediction of Applications on Simulated Systems

4.1 Overview

The power consumption of computing resources has been a significant concern in recent times. This concern is due to the cost of power generation and its effect on environmental hazards. Therefore, power-aware computing has been a constant focus in the research community. The large and complex software applications of today require a significant amount of computational ability of computer systems. The demand for growing computational ability has resulted in advancements in computer systems, which further increases power consumption.

Hardware infrastructure provided either by commercially-of-the-self (COTS) or from the cloud provides several computer systems for selection. Software applications running on computer systems with diverse hardware features have different performance and power footprints. In some cases, the performance of a given software on one system may be comparable to another system. However, the power consumption may vary due to the difference in hardware features between computer systems. Given the software, we need to select the system(s) that provides optimal performance with a smaller power requirement without running a software application. It means we need to predict the performance and the power consumption for a given software application and the system's hardware features. This problem is a multivariate performance and power prediction problem.

Researchers have proposed performance and power prediction models for computing systems. For example, work in [3] predicts performance and power of various applications using LASSO regression for one specific ARM-based target system by using performance counters collected from an x86-based system. Work in [34] has demonstrated improving the accuracy of performance and power model of gem5, and McPAT for the simulated system built based on ODROID-XU3 systems. Performance and power prediction of four heterogeneous systems using two different neural networks is shown in [4]. Work in [35] uses gem5 and McPAT tool to analyze (without predicting) the improvement in performance and power as the vector length changes for vector-oriented N-body and Triad benchmarks. Similarly, [10] uses GPGPUSim to predict the performance and power for NVIDIA' Fermi GPU with changes in hardware features. All of these research works either focuses only on the prediction of performance (runtime) or have built two separate machine learning prediction models, one to predict the performance and the other to predict the power consumption. In contrast, we proposed a method to build a single machine learning model to predict performance and power simultaneously that we called the multivariate performance and power prediction model, which is a different goal.

Work in [82] estimates performance and power consumption of workloads using six benchmark applications for big-LITTLE ARM architecture with power values obtained through estimation from performance counter values and not through measurement. Work in [18] plugs in performance and power values from an already collected large number of data points for changes in hardware features values in a heterogeneous system. Work in [83] considers both power and performance for a generalized class of parallel and distributed systems for scientific applications. These research works use estimated power values as actual for modeling purposes while we measure the power from the system to use as actual for our model.

There are research works that focus only on power models without consideration of performance. For example, power modeling is used in [84] to improve the power consumption of virtual machines reducing operational costs in

the cloud. Work in [85] uses power consumption obtained through prediction as feedback to improve power supply units used in physical computer systems. In [86] and [87] system-level power prediction is performed to understand the effect of component-level power consumption usage. Work in [33] introduces a scale-out and scale-up framework to reduce power consumption by efficient thread-level parallelism and migration in heterogeneous cores. Our work differs from these works because our goal is to predict the performance and power both and not just focus on the prediction of power consumption.

In this chapter, we provided a methodology for performance and power prediction to solve multivariate performance and power prediction problem. The four important aspects of our work are summarized here: First, we selected kernel and benchmark applications as workloads for our prediction model according to their known compute-bound and memory-bound patterns to consider them as white-box. Second, we developed a methodology to collect the actual power consumption using McPAT tool [38] which utilizes the relevant information from gem5 simulator upon execution of an application on gem5 simulated systems. Third, we built a single performance-power prediction model to perform predictions for multiple instruction-set-architecture, ARM-based, and x86-based. Finally, by categorizing the multivariate performance-power prediction problem as a multi-target regression problem rather than a multi-model problem, we trained a single machine learning model that predicted both performance and power simultaneously. Furthermore, accurate performance and power prediction of benchmark applications and kernels can be extrapolated for real applications with multiple kernels and multiple phases, as shown in [19].

The remainder of the chapter is organized as follows: Section 4.2 describes the multivariate performance and power prediction model based on machine learning algorithms that we propose. Section 4.3 describes the procedure to build the simulated systems in the gem5 simulator. It further provides information for executing selected benchmark applications on simulated systems to collect performance and system hardware features. It also describes the process of generating the power consumption for each execution using McPAT. Section 4.4 articulates

the results from learning-based models. Finally, section 4.5 has concluding remarks along with tasks that we plan to continue in future work.

4.2 Multivariate Prediction Model

We had two options for building a machine learning model for solving multivariate prediction problem. Option one was to build two different machine learning models, the first model to predict the performance and the other to predict power. Option two was to build a single machine learning model that predicts both performance and power simultaneously. A problem where multiple outputs are related to each other is known as a multi-target regression problem, as defined in [88]. We categorized the problem of performance and power prediction as a multi-target regression problem because there is a relationship between an application's performance and how much power it consumes. Therefore, we chose option two and developed a single machine learning model to predict multiple outputs, performance, and power.

Our multivariate prediction model shown in figure 4.1 has three phases; data collection phase, learning (training) phase, and prediction (testing) phase. In the data collection phase, we collected the system's hardware features and the actual performance ("runtime") by executing a chosen set of benchmark applications (such as matrix multiplication, image processing) on selected computer systems with features representing the general population of computer systems available today. We also gathered actual power consumption for each execution, as described in 4.3.2.

In the training phase, the learning model is trained for a given benchmark application using M samples of computer systems to learn the relationship between hardware features and the actual multi-target output (i.e. actual performance and power) of the same benchmark application. We denoted X_j as a set of hardware features with categorical or continuous values for j^{th} system, where $j, 1 \leq j \leq M$. We encoded features with text data and normalized the feature set X_j into $X'_j \in \mathbb{R}^d$. We collected the actual runtime and power from selected (M)

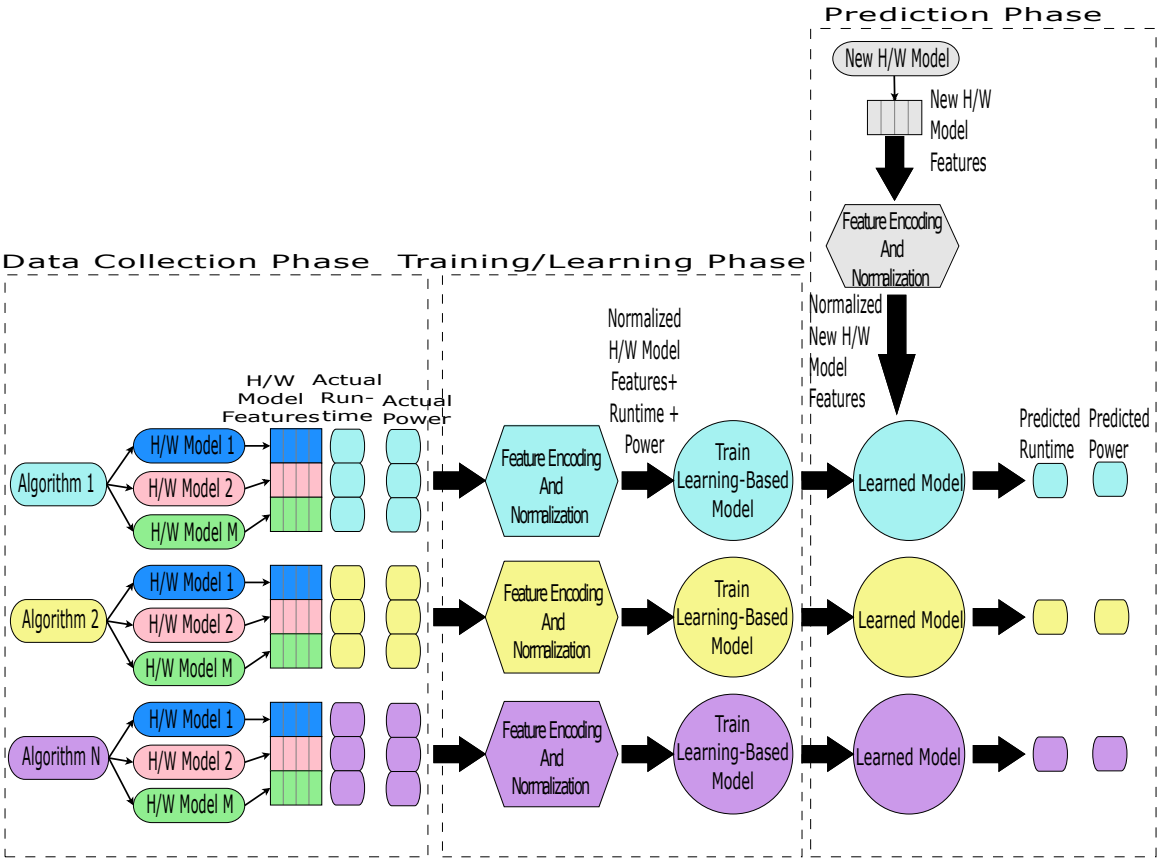


Figure 4.1: Learning-Based Performance and Power Prediction Model

training samples denoted as $Y_j \in \mathbb{R}^2$ to train the model. The training phase's objective is to find a function $\Im(X'_j) \approx Y_j \forall j$ for which we employ machine learning regression models. At the end of the training phase, the machine learning regression model learns function $\Im$ by mapping the normalized hardware features as input to the actual multi-target output of a benchmark application, which we referred to as the "Learned Model."

In the prediction phase, we selected new computer systems unseen during the training phase. Therefore, the actual runtime and power consumption of a given benchmark application for these systems were unknown. We gathered the hardware features of these new computer systems denoted as X_{new} and followed the same normalization process to get $X'_{new} \in \mathbb{R}^d$. Normalized hardware features X'_{new} of new systems were provided as an input to the "Learned Model." The learned model then predicted the multivariate output (performance and power) $Y_{new} \in \mathbb{R}^2$ for new systems using learned relationship from training

phase $\Im(X'_{new}) \rightarrow Y_{new}$ for the same benchmark application.

4.3 Experimental Setup

In this section, first, we describe the process of building gem5 simulated systems. We then explain the selection of kernel and benchmark applications for workload selection and the process to gather the actual runtime and dynamic power consumption of benchmark application execution on simulated systems built into the gem5 simulator.

4.3.1 Dataset Construction

The gem5 simulator [32] has been widely accepted simulator for architectural research. We built 475 simulated systems in the gem5 simulator, out of which 120 were based on ARM-based instruction set, and the rest were x86-based instruction set as described in chapter 2.3.1. We collected feature values from the real systems as shown in table 2.1 to use them for constructing systems in the gem5 simulator.

To test our multivariate prediction model, we selected kernels and benchmark applications as workloads according to their computation and data access patterns widely used in the real world, as shown in table 4.1. Monte carlo requires more computation than data access, categorizing it as compute-intensive (compute-bound). On the other hand, matrix multiplication requires more data access than computation categorizing it as data-intensive (memory-bound). Quicksort requires about the same amount of computation and data access, therefore, categorized as compute-plus-data-intensive (compute-plus-memory-bound). These categorization is stated in [61]. We have also used two image processing applications that perform image stitching (stitch), an image analysis application, and maximally stable regions (mser) from San Diego Vision benchmarks (SD-VBS) [70]. In addition, a network protocol application dijkstra and an encryption application sha from MiBench benchmark [63] have been used. As per SD-VBS documentation, mser is compute-bound (compute-intensive), and stitch is memory-bound (data-intensive). Similarly, dijkstra is memory-bound, whereas

Table 4.1: Applications used as workloads for Multivariate Model

Application	Benchmark	Intensiveness
monte carlo		compute-intensive
matrix multiplication		data-intensive
quiksort	MiBench	compute-intensive and data-intensive
mser	SD-VBS	compute-intensive
stitch	SD-VBS	data-intensive
sha	MiBench	compute-intensive
dijkstra	MiBench	data-intensive

compute-intensive = compute-bound
data-intensive = memory-bound

sha is compute-bound.

Each of these benchmark applications with different problem sizes [Problem Size (PS) = $\Im(Application)$] were executed on each of the 475 simulated systems resulting in the "475 x NoOfApps x PS" sample size of the dataset. We collected performance (actual runtime) values for each execution. Collecting power consumption for each execution was not as trivial, so we developed a separate process to collect dynamic runtime power consumed, as described in section 4.3.2. This process was carried out for each of the samples in the dataset.

4.3.2 Collecting Dynamic Runtime Power

McPAT tool is built for integrated power, area, and timing modeling. It accepts the XML file as an input with gem5 simulated system's configurations and statistics of application execution. Using the information from an XML file, McPAT generates dynamic power consumption at the hardware component level such as processor, cache and memory, and overall system level. We leveraged this McPAT's capability to gather power consumption for our multivariate prediction model. Figure 4.2 shows the process flow for collecting dynamic power consumption using McPAT.

During each execution of the benchmark application on gem5 simulated systems [475 (simulated systems) x NoOfApps x PS], a set of configuration and statistic files was generated. The configuration file has simulated system hardware architecture features such as instruction set, cores, cache, memory controller. The

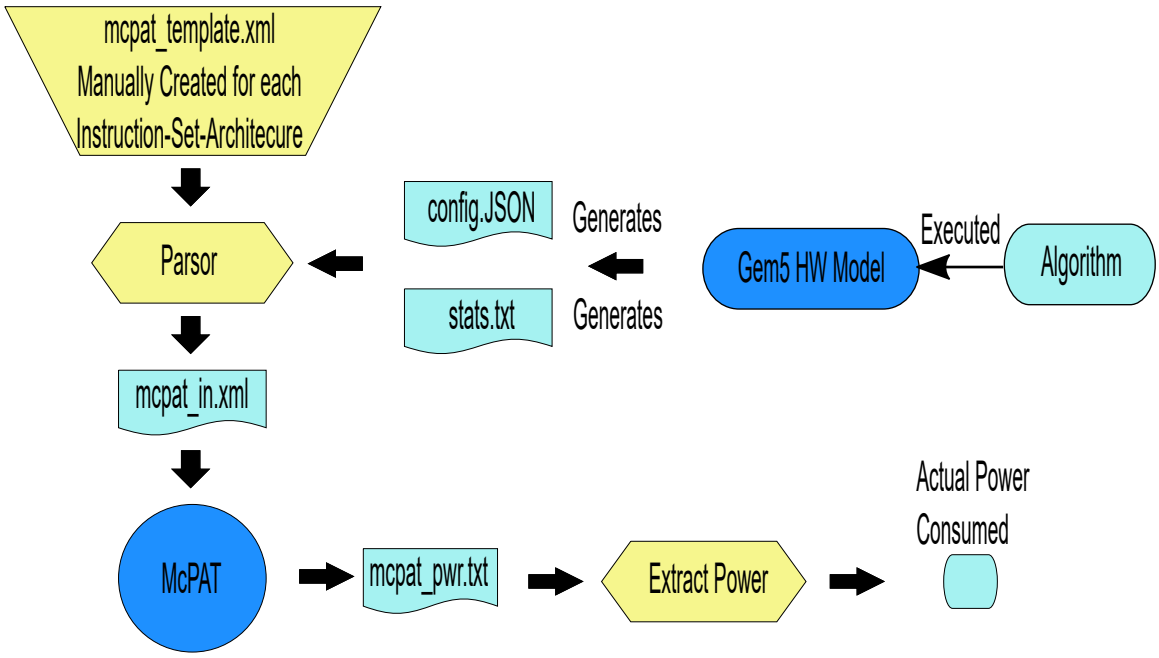


Figure 4.2: Power Generation & Collection Process for Simulated Systems

statistics file has execution statistics of a benchmark application on the simulated system indicating usage of each hardware component. For example, the number of reads from cache or memory, number of writes in cache or memory, number of reads and write misses from cache/memory, etc.

The parser program produces McPAT input XML by mapping the fields from gem5 generated configuration and statistic files into template XML. The McPAT template XML files were manually generated for each architecture (x86 and ARM) because the fields in configuration files and statistics files may vary depending on the simulated system's hardware features. We executed a parser program for each set of configuration and statistics files to generate McPAT input XML using the template XML. We then fed each of the input XML files to the McPAT tool to generate the dynamic power consumption.

4.3.3 Training and Prediction Model

We have used the decision tree regression model [57] from scikit-learn, a python-based machine learning library [65] for multivariate performance and power prediction. Given the input features x_i from input feature set $x = x_0, x_1, ..., x_n$ and target value y, decision tree regression model recursively partitions feature space

of x_i such that samples with close target value are on the same side of the tree. For regression problems, mean squared error (MSE) is used to identify the closeness between actual and target values and is given by:

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \bar{y}_i)^2 \tag{4.3.1}$$

Let data of node m is represented by Q. At each of the decision tree node m, there could be several possible partitions in dataset Q. For each candidate partition $\theta = (x_i, t_{x_i})$, data is partitioned into $Q_{left}(\theta)$ and $Q_{right}(\theta)$ (equation 4.3.2), where x_i is compared against the threshold t_m.

$$Q_{left}(\theta) = (x, y)|x_i <= t_m \tag{4.3.2a}$$

$$Q_{right}(\theta) = Q \setminus Q_{left}(\theta) \tag{4.3.2b}$$

For each of the candidate partitions, mean squared error (MSE) is calculated for the left and the right partitions which is then used to calculate the impurity function given by

$$G(Q, \theta) = \frac{n_{left}}{N_m} MSE(Q_{left}(\theta)) + \frac{n_{right}}{N_m} MSE(Q_{right}(\theta)) \tag{4.3.3}$$

Out of all the candidate partitions, select the one which minimizes the impurity

$$\theta^* = argmin_{\theta(x_m, t_m)}\, G(Q, \theta(x_i, t_i)) \tag{4.3.4}$$

At the root node, candidate partitions consist of one partition for each input feature $x_i|x_i \in x$ input feature set and threshold t_{x_i}. For each of the candidate partitions, impurity function $G(Q, \theta)$ is calculated using mean squared error. Partition with smallest impurity is selected as criteria $\theta = (X_i, t_m)$ for this node. Based on criteria $\theta = (X_i, t_m)$, all the samples Q from root node with $x_i <= t_m$ are partitioned into $Q_{left}(\theta)$ and $x_i > t_m$ are partitioned into $Q_{right}(\theta)$. Recursively partition the subsets $Q_{left}(\theta)$ and $Q_{right}(\theta)$ until maximum allowable depth of the tree is reached $N_m < min_{samples}$ or sample size has reached to one $N_m = 1$, that

Table 4.2: Prediction Accuracy for Different Train-Test Ratio

	Runtime			Power		
Application	**40:60**	**50:50**	**60:40**	**40:60**	**50:50**	**60:40**
monte carlo	0.15	0.13	0.01	0.049	0.16	0.014
matrix multiplication	6.15	2.29	1.59	0.57	0.52	0.32
quiksort	11.84	4.6	1.44	0.77	0.83	0.001
mser	74.09	7.03	0.0005	1.72	1.95	0.0020
stitch	5.17	1.35	0.45	6.96	1.83	2.5
sha	3.44	0.3	0.26	2.89	0.9	0.04
dijkstra	13.56	0.62	0.18	11.82	0.1	0.025

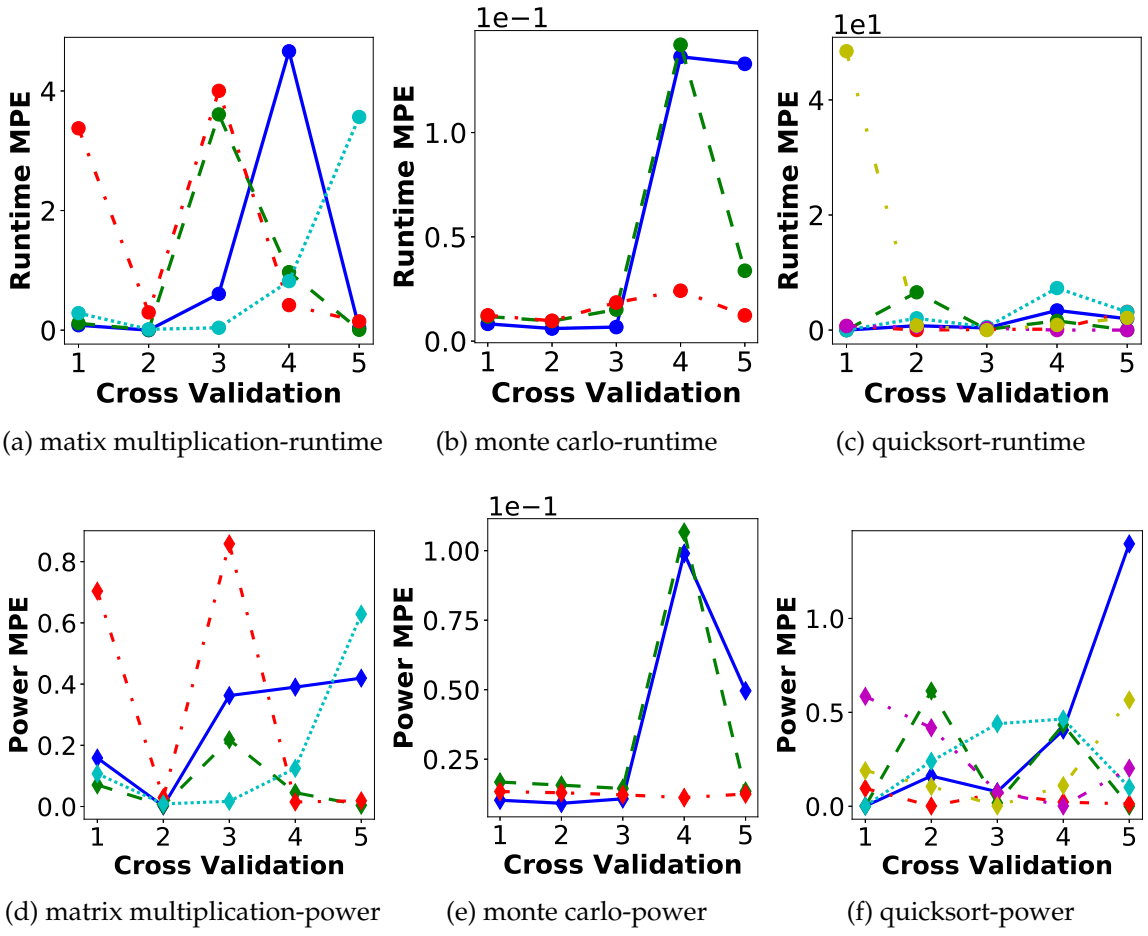
means, mean squared error cannot be reduced further. Each of the non-leaf nodes are the decision points that guide the direction in which the data path is traversing for a set of specific feature values of a sample until a leaf node is reached which provides the target (predicted) value y for that sample. Mean of y values is considered as the target value for the leaf node with more than one samples.

4.4 Results

We collected runtime by executing selected benchmark applications with different problem sizes on all 475 gem5 simulated systems and the hardware features of gem5 simulated systems. Power consumption for each execution was collected using the process described in section 4.3.2. We split the dataset into 60:40 train-test set ratio using train_test_split() function from scikit-learn which internally calls ShuffleSplit() function to randomize the samples selection. We then trained the decision tree regression model using a 60% training dataset consisting of the hardware features, runtime, and power consumption for each benchmark application. The train-test split ratio of 60:40 was selected since the 60:40 split had better prediction accuracy for all benchmark applications, as shown in table 4.2. In the prediction phase, the trained decision tree model was provided only the hardware features of the remaining 40% test simulated systems as an input, unseen during the training phase, to predict both the runtime and power consumption values. We evaluated our multivariate performance and power prediction model statistically first using cross-validation and then the overall accuracy of the approach.

4.4.1 Cross-Validation Error

We performed five-fold cross-validation [80] using 60% of dataset samples used for training. Figure 4.3 shows runtime and power cross-validation mean percentage error (MPE) for each fold separately for matrix multiplication, monte carlo and quicksort. We calculated the median of MPEs for each problem size for a given application which we label as MedPE. For all five folds, matrix multiplication has average runtime MedPE of 1.15, power MedPE of 0.2, monte carlo has average runtime MedPE of 0.038, and power MedPE of 0.027 and quicksort has average runtime MedPE of 2.86 and power MedPE of 0.22.



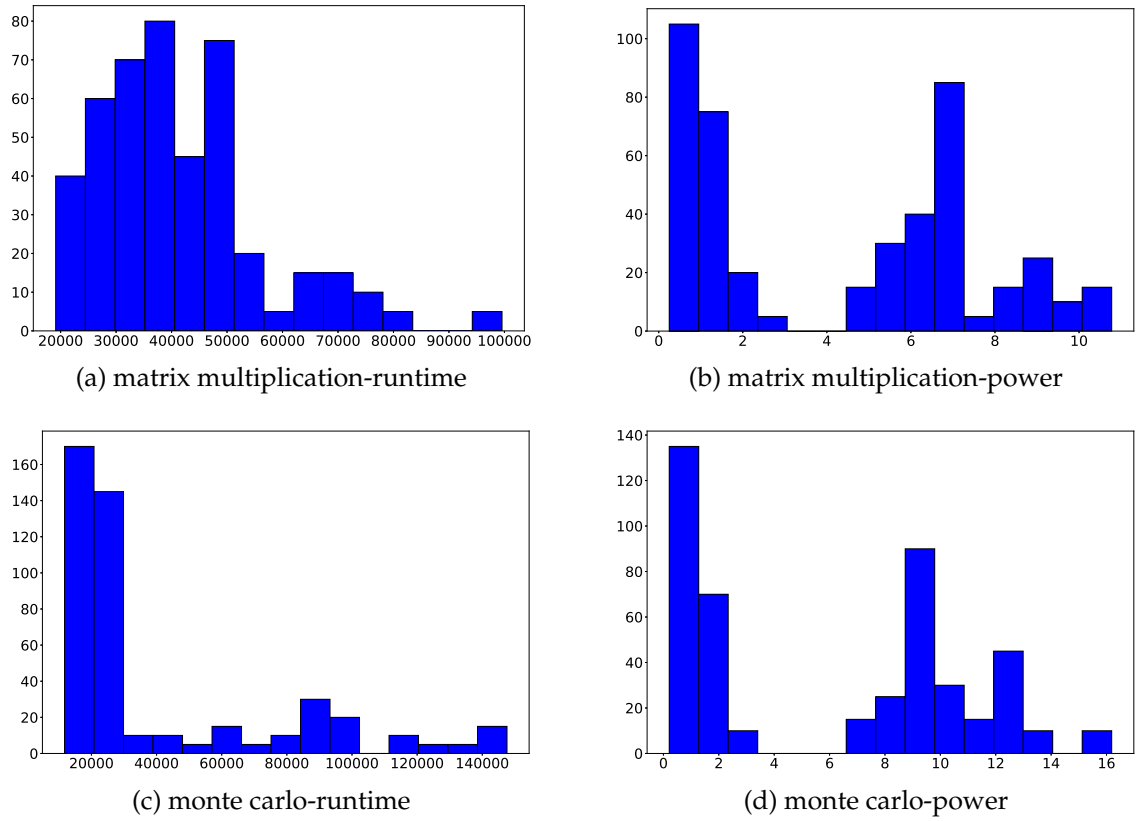
(a) matix multiplication-runtime (b) monte carlo-runtime (c) quicksort-runtime

(d) matrix multiplication-power (e) monte carlo-power (f) quicksort-power

Figure 4.3: Cross-Validation Prediction Error Per Problem Size

We observed that for memory-bound matrix multiplication, MedPE for runtime is an order of magnitude higher than runtime MedPE for compute-bound monte carlo. Additionally, power MedPE is lower than runtime MedPE for both

applications. We believe that different MedPE for runtime and power is due to the difference in the distributions of samples over the runtime and power values range. Figure 4.4 shows sample distributions of runtime and power for matrix multiplication and monte carlo applications. We infer from this histogram that samples of matrix multiplication are widely spread across a range of runtime values between 20000 to 50000, whereas the power values of one, two, and seven dominate the sample distribution. On the other hand, for monte carlo, the majority of samples have runtime between 20000 to 25000 and power values of one and nine having a narrow spread.



(a) matrix multiplication-runtime

(b) matrix multiplication-power

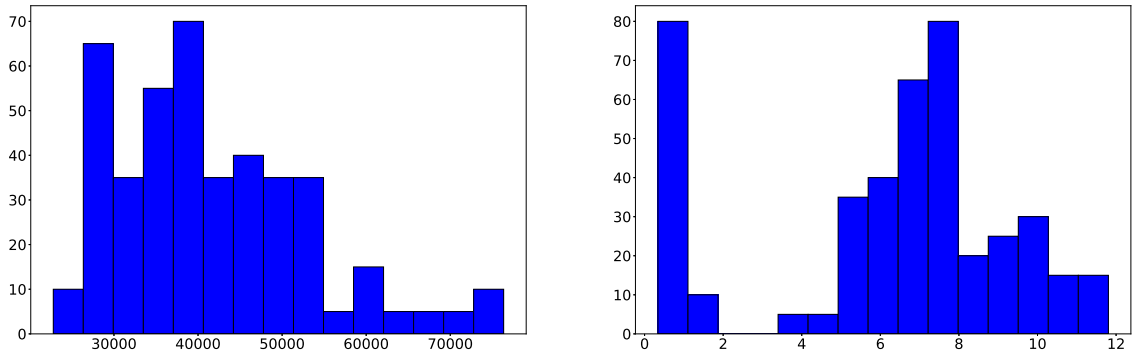
(c) monte carlo-runtime

(d) monte carlo-power

Figure 4.4: Samples Distribution over Runtime vs Power

We observed similar behavior in other benchmark applications from the histogram in figure 4.5. A memory-bound stitch (ST) application, has a wider spread for runtime as compared to mser (MS), a compute-bound application resulting in higher runtime median percentage error for stitch. Similarly, dijkstra (DI), a memory-bound application, has a higher runtime median percentage error than compute-bound sha (SH). We also see that the median percentage error for power

is less than runtime for stitch, dijkstra, and sha because power has a narrow spread as compare to runtime for each of these three benchmark applications.



(a) stitch-runtime (b) stitch-power

(c) mser-runtime (d) mser-power

Figure 4.5: Samples Distribution over Runtime vs Power for Benchmarks

4.4.2 Prediction Accuracy

To demonstrate the overall accuracy of our approach, we predicted runtime and power simultaneously for 40% dataset test systems samples using a single multivariate prediction model trained from 60% dataset train systems samples. Figure 4.6 shows the median percentage error (MedPE) prediction error for test systems. Matrix multiplication has average runtime MedPE of 0.29, power MedPE of 0.089, monte carlo has average runtime MedPE of 0.13, and power MedPE of 0.13 and quicksort has average runtime MedPE of 0.64 and power MedPE of 0.10. We observed that the compute-bound monte carlo application has lower runtime median percentage errors as compared to memory-bound matrix multiplication and both-bound quicksort. Also, the power error is generally lower than the corre-

sponding runtime error.

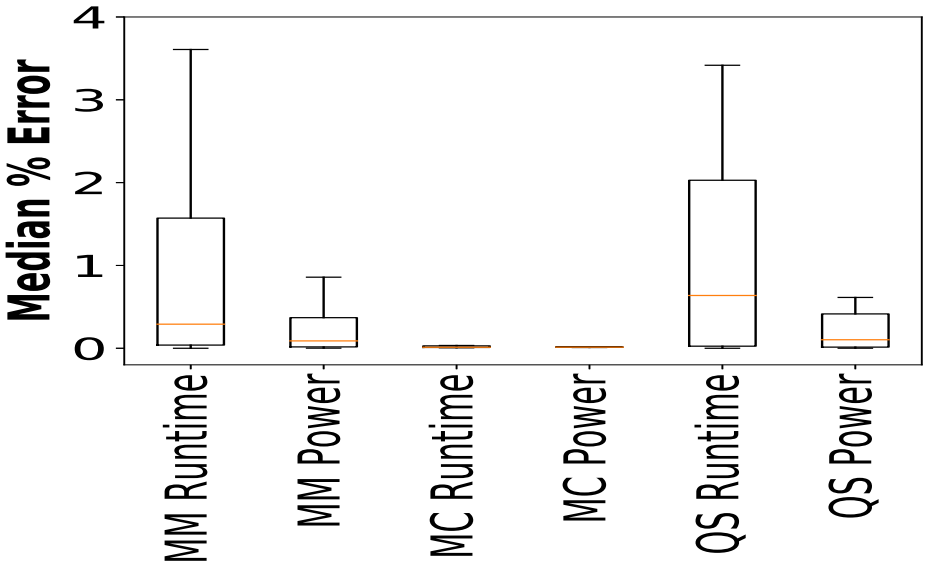


Figure 4.6: Prediction Accuracy for Test Simulated Systems

To understand the reasons behind these observations, we further analyzed the results. The processor features have a higher impact on the performance (runtime) of a compute-bound application. On the other hand, memory features dominate the runtime of a memory-bound application. The gem5 simulator supports only one superscalar processor with five stages of out-of-order pipeline that we have used to build all of the simulated systems. It causes the compute-bound monte carlo application to have lower variations in runtime values from simulated systems. Whereas, the gem5 simulator supports several different memory resulting in higher variations in the runtime of the memory-bound application, such as matrix multiplication. Due to higher runtime variations in matrix multiplication, the runtime prediction error is higher as compared to monte carlo. Quicksort application depends on both a processor and memory features; therefore, error for quicksort is similar to matrix multiplication. We observed that the power error is less than runtime error for all the three benchmark applications, whether compute-bound or memory-bound because the memory unit consumes much less power than the computation unit resulting in lower variance and lower error for power for all applications.

We made similar observations for benchmark programs from figure 4.7 with

the stitch (ST) having runtime MedPE of 0.41 and power MedPE of 0.041, mser (MS) having runtime MedPE of 0.34 and power MedPE of 0.34, dijkstra (DT) runtime MedPE of 1.88 and power MedPE of 0.44 and finally sha with runtime MedPE of 0.32 and power MedPE of 0.18. Here, runtime error for dikstra (DI) is higher as compared to sha (SH) because a memory-bound dikstra has higher dependence on memory features resulting in higher variance in runtime for different memory units. In comparison, a compute-bound sha has lower runtime variance with the use of the same processor model for building all gem5 simulated systems. Once again, the power error for all four benchmark applications is lower than the corresponding runtime error because of the smaller contribution of power consumption of different memory units of gem5 simulated systems as compared to the computing unit.

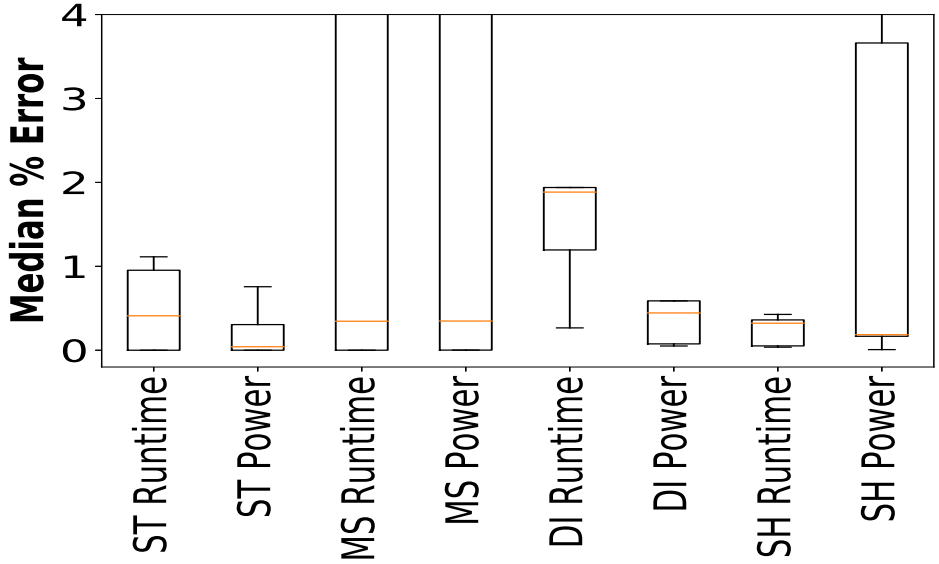


Figure 4.7: Prediction Accuracy of Test Simulated Systems for Benchmarks

4.5 Summary

This chapter proposed a learning-based approach to multivariate performance and power prediction. In this approach, different simulated systems were characterized according to their key processor, cache, and memory features to build gem5 simulated systems. To collect actual power consumption in addition to the

performance of a software execution on the gem5 simulated systems, we formulated a process using the McPAT tool. We demonstrated that a single learning-based prediction model predicts both performance and power consumption simultaneously with high accuracy of greater than 95% for different software applications. We established that our prediction model worked even with vastly different architectures such as x86 and ARM. Our results showed that for gem5 simulated systems, due to higher runtime variations, power prediction accuracy is higher than in performance. A simple gem5 processor design results in lower performance errors in compute-bound applications compared to memory-bound. In this work, benchmark applications used have serial code executed on the gem5 simulator but in the future, we plan to extend our model for parallel benchmarks on physical systems to test the model's adaptability for workload variations. In the future, we aim to perform multivariate predictions on physical computer systems.

CHAPTER 5

Cross Prediction with Scaling using Machine Learning

5.1 Performance Prediction of Physical Computer Systems Using Simulated Systems

5.1.1 Overview

The progression in computer systems architecture has made it possible to have several options for the selection of computer systems available today. A software performance will differ on each computer system according to its architectural (hardware) features. To obtain the performance of a software application on a specific computer system, we need to acquire the system. However, it is an exhaustive task to procure several computer systems just to collect the performance of a software application to evaluate them for selection. On the other hand, several simulated systems can be built easily in a short duration with simulation tools to evaluate the performance of the software. Therefore, we need to use simulated systems to predict the performance of the physical systems, which is a difficult problem to solve.

Performance prediction has been an active research topic. The GPU performance is predicted using features collected during the execution of an application on the x86-based system in [20] and [51]. For unseen software, a performance on an ARM-based system is predicted in [50] by collecting performance counters on an x86-based system. Work in [5] shows that a trained machine-learning

model from one platform can be retrained only with one percent of samples from test (another) platform to predict ninety-nine percent of performance data. These works have shown that a machine-learning model trained on performance data from one physical system can predict the performance of another physical system with a disparate set of hardware features, which is a different goal.

Cycle-accurate simulators [89] are widely used for evaluating software performance with good accuracy. One of which is gem5 simulator [32], a powerful well-accepted cycle-accurate simulator. The gem5 simulator provides two modes for simulating systems; full-system mode and system-call emulation mode. Work in [31] has shown that gem5 simulated system built with full-system mode provides an accurate software performance compared to the physical system. However, full-system set up for each system is arduous and slow as compared to native execution [39] [40]. On the other hand, the system-call emulation mode of gem5 provides systems simulation at a higher speed with lower accuracy. Therefore, our goal is to, perform accurate performance prediction of physical systems using gem5 simulated systems constructed using system-call emulation mode.

In this work, we performed "Cross Performance Prediction", in which, a machine-learning model is trained only on gem5 simulated systems, built using system-call emulation mode, to predict the performance of physical systems. We evaluated cross performance prediction models for MiBench [63] and San Diego Vision Benchmark Suite (SD-VBS) [70] applications from different application domains having different computation and memory access patterns.

The remainder of the section is organized as follows: Section 5.1.2 describes the cross performance prediction model. Section 5.1.3 describes the procedure to build simulated systems, applications selection for workload, and physical systems selection for cross performance prediction. Section 5.1.4 articulates the results from the cross performance prediction model. Finally, section 5.1.5 has concluding remarks along with tasks that we plan to continue in future work.

5.1.2 Cross Performance Prediction Model

Our cross performance prediction model shown in figure 5.1 has three phases; training, cross performance prediction and model evaluation. In the training phase, the machine-learning regression model is trained for a given application to learn the relationship between the system's hardware features and the actual simulator performance ("actual runtime") of the same application. We selected sample systems (M) for the training phase with hardware feature values that represents the general population of systems available today denoted as Xs_j where $j, 1 \leq j \leq M$. We encoded hardware features with text data from Xs_j into $Xs'_j \in \mathbb{R}^d$. For example, systems with nine features represented as $Xs'_j \in \mathbb{R}^9$. We represented the actual simulator runtime of selected (M) samples for a given application as $ys_j \in \mathbb{R}$ in the training phase to train the model. The objective of the training phase is to find a function $\Im(Xs'_j) \approx ys_j \forall j$, the mapping between systems' hardware features and actual simulator runtimes for an application, which we referred to as "Learned Model".

In the cross performance prediction phase; first, we collected hardware features Xp_i from each of the (N) physical systems $i, 1 \leq i \leq N$. The hardware features with textual data were then encoded from Xp_i into $Xp'_i \in \mathbb{R}^d$. These physical systems' hardware features Xp'_i were provided as an input to an application-specific learned model to predict the runtime $yspred_i$ for physical computer systems using learned relationship from training phase $\Im(Xp'_i) \rightarrow yspred_i$.

In the model evaluation phase, we validated the predicted runtime $yspred_i$ from the cross prediction model to the actual runtime yp_i collected by executing an application on physical systems (N). To evaluate the accuracy of the learned model, we collected runtime ysp_i by executing an application on gem5 simulated systems built with the hardware features of the physical systems for which cross prediction is required and compare ysp_i runtime with $yspred_i$. Runtimes ysp_i and cross predicted runtime $yspred_i$ will have a difference of a factor with the runtime from physical system yp_i, we labeled it as a "Scaling Factor". We determined scaling factor depending upon the type of application and was applied to ysp_i and $yspred_i$ to improve the accuracy of the model.

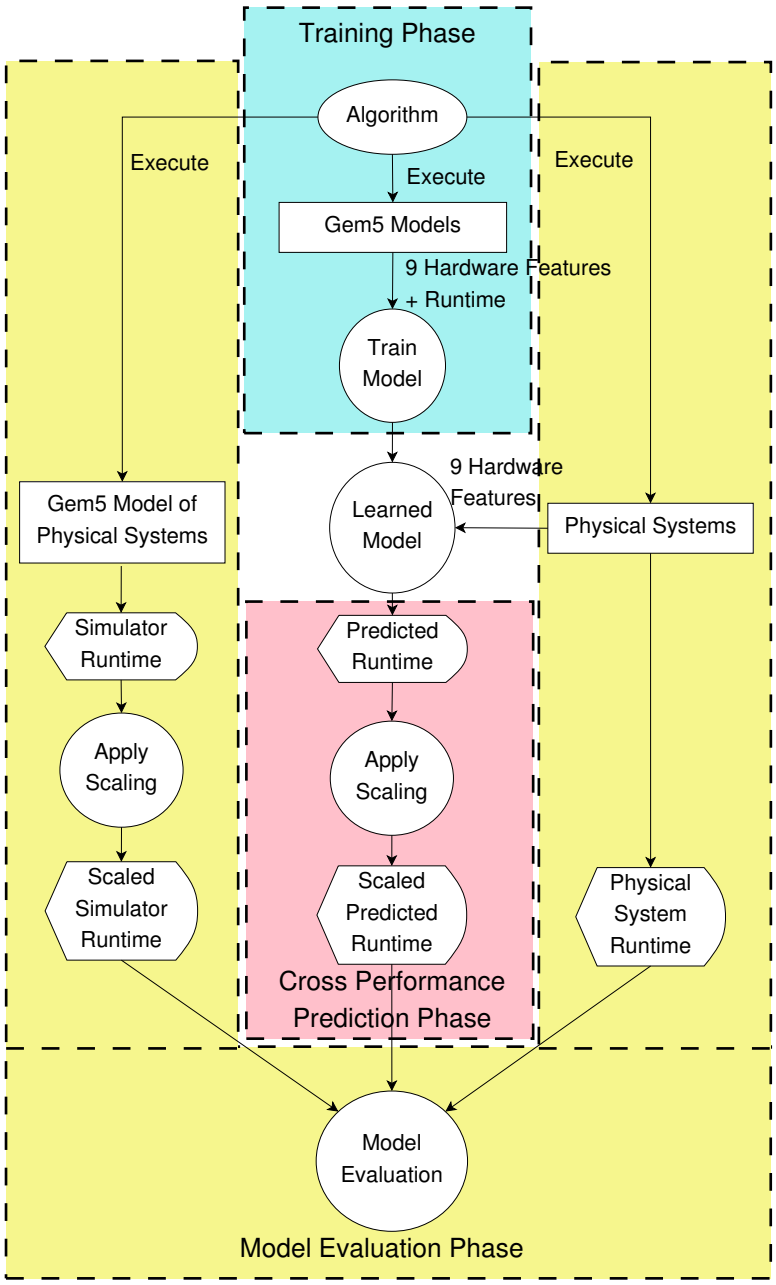


Figure 5.1: Cross Performance Prediction Framework

5.1.3 Experimental Setup

In this section, we provide detail of building gem5 simulated systems, articulate application selection for workloads and then provide information about physical systems used for cross performance prediction.

Table 5.1: Applications used as workloads for Cross Performance Prediction Model with Scaling

Application	Benchmark	Intensiveness
mser	SD-VBS	compute-intensive
svm	SD-VBS	compute-intensive
tracking	SD-VBS	data-intensive
stitch	SD-VBS	compute-intensive and data-intensive
dijkstra	MiBench	data-intensive
sha	MiBench	compute-intensive

compute-intensive = compute-bound
data-intensive = memory-bound

5.1.3.1 Simulation-based Models for Training

To show the effectiveness of our cross performance prediction model for the real-world systems, we collected hardware features from the computer systems available in the market today as shown in table 2.1. Using these features, we built 475 systems in the gem5 simulator using system-call emulation mode as described in chapter 2.3.1. Due to the limitations and differences in the design of gem5 simulated systems compared to physical systems presented in chapter 2.3.1, simulated systems are a close representation but not the exact replica of physical systems.

5.1.3.2 Applications Selection for Workload

We selected four applications mser, svm, tracking and stitch from San Diego Vision Benchmark (SD-VBS) and two applications sha and dijkstra from MiBench benchmark each representing an application from the different domain as shown in table 5.1. According to [70], mser and svm are compute-bound, tracking is memory-bound whereas stitch is compute-plus-memory-bound. Work in [11] depicts that the correlation between hardware features and runtime can be used to understand the hardware features that affect runtime the most. In this way, we can identify whether processor features or memory features dominate the runtime categorizing application into compute-intensive (compute-bound) or data-intensive (memory-bound) respectively.

We plotted a pearson correlation coefficient [69] of nine hardware features with runtime for each of the six applications in figure 5.2. For mser, processor feature

L2 cache size has a higher correlation with runtime than memory features categorizing mser as compute-bound (or compute-intensive). Similarly, for svm, a compute-bound application, has a higher correlation for L1 and L2 cache sizes with runtime. A memory-bound (or data-intensive) application tracking has a higher correlation between memory features (memory clock speed and types) with runtime. Even though MiBench does not categorize dijkstra and sha applications, we categorized dijkstra to be data-intensive and sha to be compute-intensive using a correlation coefficient plot. We observed that sha has a marginally higher correlation of L2 cache size compared to memory features because sha reads data to be encrypted from a file causing it to also have some dependence on data.

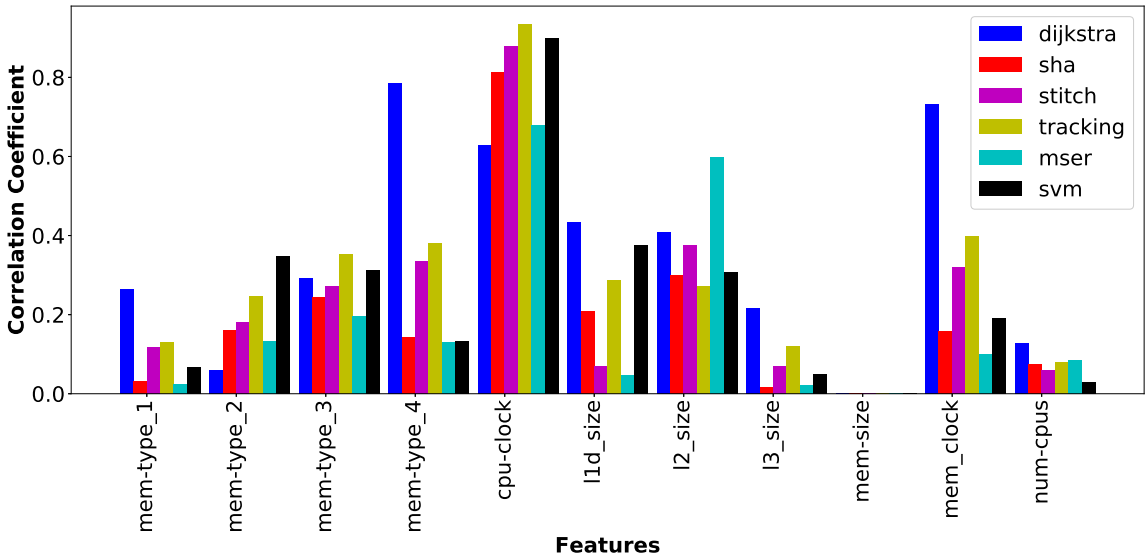


Figure 5.2: Pearson Correlation Coefficient of Hardware Features with Runtime

5.1.3.3 Physical Computer Systems for Cross Performance Prediction

For cross performance prediction experiments, we kept our scope limited to x86-based computer systems mainly using Intel processors. We considered physical systems with varied values of all nine hardware features as shown in table 5.2. The selection of these systems consists of a server system with Intel Xeon E5-2620 v3 processor, a high-end system having Intel Core i7-8700 processor and the general-purpose systems with Intel Core i7-6500U and Intel Core i5-6500 processors with

different memory types and sizes. We collected the values of these nine hardware features from physical systems using the dmidecode utility.

Table 5.2: Physical Computer Systems for Cross Performance Prediction Model with Scaling

Sr	ISA	CPU Speed GHz	Cores	Mem Type	Mem Access MHz	Mem Size GB	L1-L3 Cache Size
A	x86	3.2	4	DDR3	1600	4	32,6
B	x86	2.4	12	DDR4	1866	16	32,15
C	x86	3	2	DDR3	1600	8	32,4
D	x86	3.2	6	DDR4	2666	16	32,12

**L1 Cache Size is in kB and L3 Cache Size is in MB
Configuration taken from following models
A.Intel Core i5-6500 B.Intel Xeon E5-2620 v3 C.Intel Core i7-6500U D.Intel Core i7-8700

5.1.4 Results

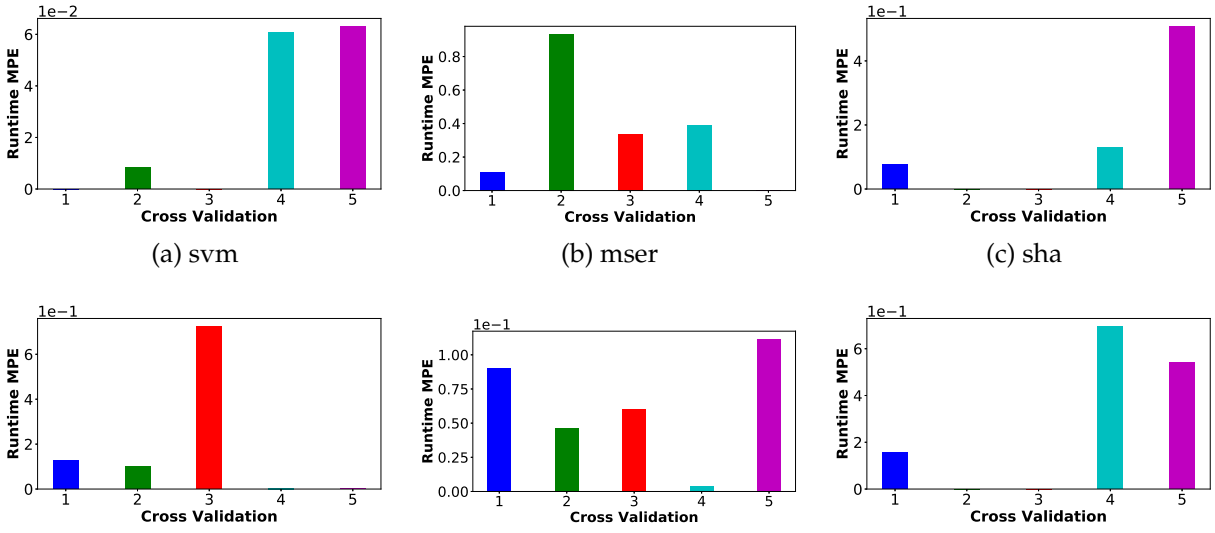
We executed each of the applications from table 5.1 on each of the 475 gem5 simulated systems from table 2.1 and collected runtime for each execution. We collected nine hardware features $Xs_j \in \mathbb{R}^9$ and runtime $ys_j \in \mathbb{R}$ where j represents all 475 simulated systems. Two of the hardware features with text data instruction-set-architecture (ISA) and memory-type were then converted to real-value using encoding technique resulting in hardware features with all real-valued dataset $Xs'_j \in \mathbb{R}^9$. Therefore, the performance dataset consists of records with nine real-valued hardware features and the corresponding runtime $[Xs'_j, ys_j]$ for an application execution from each simulated system.

For training and prediction, we used the decision tree regression machine-learning model [90] [6] from a python-based library scikit-learn [65]. We randomly selected 60% of $[Xs'_j, ys_j]$ performance records using the SuffleSplit() function from scikit-learn and trained the decision tree model for each application separately which is called the "Learned Model." The learned model was then provided encoded nine features from physical systems $Xp'_i \in \mathbb{R}^9$ (i=1 to 4 in our case since we have four physical systems) as an input to predict the performance, that is, we performed cross performance prediction. In this section, we evaluate our

model in two ways: first, the accuracy of the learned model, that is, a decision tree model trained only on gem5 simulated systems' performance dataset is verified by performing cross-validation. Second, we evaluate the learned model for cross performance prediction by predicting runtime for x86-based physical computer systems shown in table 5.2.

5.1.4.1 Simulation-based Performance Prediction Accuracy

To evaluate the performance of the simulation-based performance prediction model, we performed five-fold cross-validation [80]. In which we trained the decision tree regression model on 60% of 475 simulated systems performance dataset $[Xs'_j, ys_j]$ and predicted the runtime for the remaining 40% (test systems) for each application separately. To show that our model provides accurate predictions for any random selection of samples from the performance dataset for training and prediction, we use ShuffleSplit() function to randomly select samples for training and prediction in each of the five folds. The mean percentage error (MPE) of the 40% samples of test systems for each of the folds is shown in figure 5.3. Our results demonstrated that the learned model achieves a performance prediction accuracy of 99% in each fold for all applications considering prediction for test simulated systems.



(a) svm (b) mser (c) sha

(d) tracking (e) stitch (f) dijkstra

Figure 5.3: Cross-Validation Mean Percentage Error

5.1.4.2 Cross Performance Prediction Accuracy

We used a decision tree regression model trained only on gem5 simulated systems' performance dataset (Learned Model) for each of the applications to predict the runtime for the physical systems from table 5.2, which we call "cross performance prediction".

To demonstrate the accuracy of cross performance prediction, we performed two tests. In the first test, we compared the cross performance predicted runtime with the actual runtime from gem5 simulated systems that were built with the same hardware feature values of nine features from physical systems. The results of the first test are shown in figure 5.4 with the red bar indicating actual runtime collected from gem5 simulated systems build from physical systems' hardware features and the blue bar indicating cross predicted runtime. Mean percentage error (MPE) considering all four physical systems for svm, mser, tracking, stitch, sha, and dijkstra are 2.29%, 8.13%, 3.87%, 1.76%, 10.86%, and 9.98% respectively. Hence, we achieved an accuracy of more than 90% considering the MPE of all four physical systems for all applications combined.

In the second test, we compared the actual runtime from physical systems to cross predicted runtime predicted from the learned model trained only on simulated systems. The results from the second test are shown in figure 5.5 with the actual runtime from physical systems in the red bar and cross predicted runtime (unscaled) in the black bar. The first observation from the plot is that server-like systems (B and D) have larger errors between cross predicted runtime and the actual runtime as compared to the general-purpose systems (A and C). This is because simulated systems are representative of the general class of machines, therefore, the difference in hardware features between the simulated system and the server-like system is larger, resulting in a larger difference in runtime.

The second observation is that when we compared actual runtimes from physical systems and the corresponding unscaled cross predicted runtimes, errors in compute-bound applications are higher than memory-bound applications. For example, sha has an error of 35.16%, whereas tracking has an error of about 10% for general-purpose systems (A and C), we call this a "Scaling Factor". This is

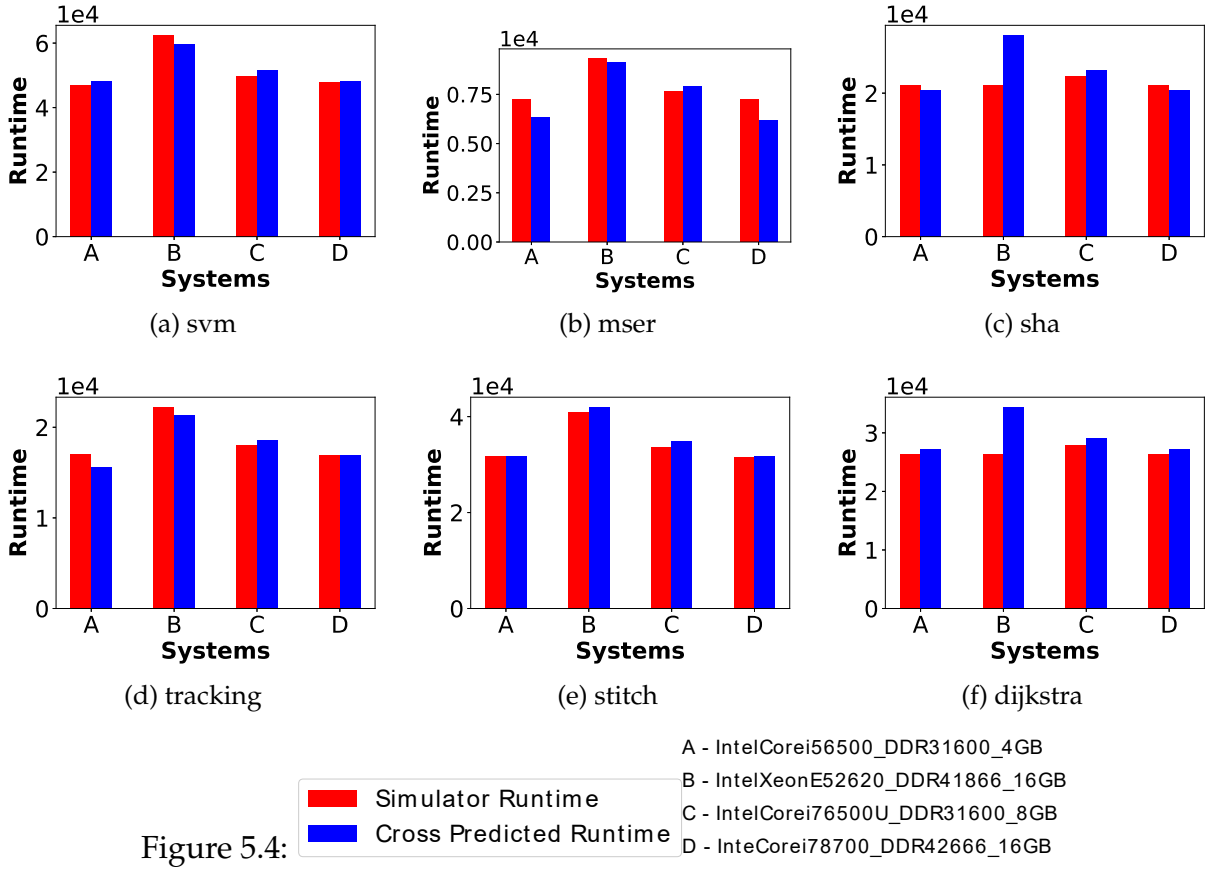


Figure 5.4: Simulator Runtime vs Cross Performance Prediction Runtime

because the gem5 simulated systems, built with the only supported superscalar processor, witness a larger gap in features with that of several disparate processors used in physical systems. This results in higher variations in runtime for compute-bound applications between simulated systems and physical systems. The higher variation in the runtime of compute-bound applications causes larger errors. On the other hand, we have the support of several different memory modules (types, clock speeds) in gem5, which enables the simulated systems to closely represent memory units of physical systems resulting in a narrow difference in runtimes for memory-bound applications between gem5 simulated systems and the physical systems. The difference between the gem5 processor and the physical system's processors exists for memory-bound applications also, but it is hidden due to higher dependence on memory which is slower as compared to a processor.

Based on the second observation, we applied the scaling factor of 35% and 10% to cross predicted runtime values of compute-bound and memory-bound

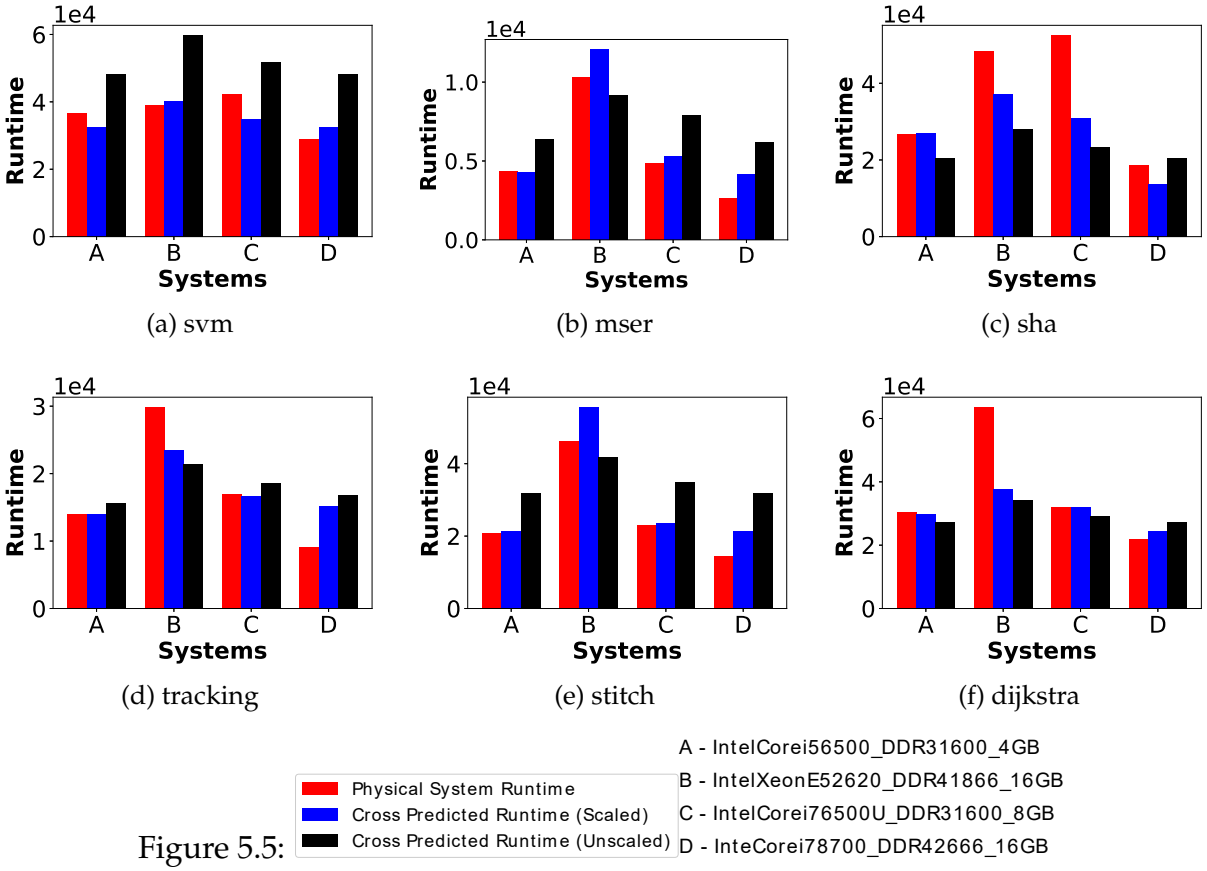


Figure 5.5: Physical System Runtime vs Cross Performance Prediction Runtime

applications, respectively. Both unscaled and scaled cross performance prediction runtime values are shown in figure 5.5. We can confirm from the plot that the error between the actual runtime and unscaled cross predicted runtime is higher compared to the scaled cross predicted runtime. The results demonstrated that by applying the scaling factor, we reduced errors in cross predicted runtimes for each application in the range of 10% to 20%. Furthermore, considering all applications combined, the scaled predicted runtime error is 2.8% and 12% for the general-purpose systems A and C, and 21% and 37% for the server systems B and D.

5.1.5 Summary

In this section, we applied cross performance prediction on benchmarks applications from MiBench and SD-VBS. First, with the help of the correlation coefficient, we categorized applications in compute-bound or memory-bound. The results demonstrated that compute-bound applications have a higher difference

in runtime between the simulated system and physical as compared to memory-bound applications. Using this information, we applied the scaling factor of 35% for compute-bound applications and 10% for memory-bound applications. After applying the scaling factor, our cross performance prediction achieved an average accuracy of 80% to 90%. In the future, we would like to extend our work to include other instruction-set architecture systems such as ARM-based physical systems. We would also like to include additional variables such as power in addition to performance for improving computer system selection using power-aware computing.

5.2 Predicting Physical Computer Systems Performance and Power from Simulated Systems using Machine Learning Model

5.2.1 Overview

Technological advancements in hardware features of computer systems have provided us with a plethora of options. In particular, features of processors, caches and memories have seen tremendous growth. The software application will have dissimilar performance and power consumption when executed on computer systems with the diversity of processor, cache and memory features. Therefore, selecting a computer system with the appropriate hardware features that provide optimal performance and power is a challenging but important problem to address. To collect the performance and power consumption of application execution on computer systems require access to physical computer systems. However, making many physical systems available just for collecting performance and power for selection is an exhaustive task. Thus, we needed a solution to accurately estimate the performance and power of software applications for computer systems with a specific processor, cache and memory features without the availability of physical systems to execute an application. This task is even more complex because physical systems are in general non-terminating and non-deterministic,

whereas the behavior of the application is terminating, deterministic and platform-independent [27]. This makes the problem of computer systems selection very difficult.

Performance has always been an important factor for the selection of computer systems but since the past decade, power consumption has emerged as a major concern due to heat profile causing environmental hazards. Furthermore, the development of complex software applications requires plentiful computer resources resulting in increased power consumption. Therefore, power-aware computing has become an active research area as discussed in [91]. The work in [84] proposes cost estimation for a cloud-based virtual machine using power prediction. Similarly, power consumption is reduced using thread-level parallelism in [33]. A study of component-level power to reduce the system-wide power consumption is the focus of work in [86] and [87]. These research works have focused primarily on power estimation without performance consideration. In contrast, our goal is to utilize both the performance and power consumption to address the computer systems selection problem.

Some research works have focused on cross-platform prediction. The work in [20] predicts GPU performance by collecting features from the x86-based system. The work in [5] builds a machine learning model from the performance dataset of one HPC system to predict performance for another HPC system. Both [20] and [5] focuses only on performance prediction without power consideration. On the other hand, the research works in [50] and [4] built machine learning models for performance and power prediction for the ARM-based target system by collecting performance counters on x86-based host systems. These works constructed two separate machine learning models for performance and power; however, due to the relationship between performance and power, we consider performance and power prediction as a multi-target prediction problem requiring a single model to predict both performance and power simultaneously as discussed in [92]. Furthermore, [20], [5], [50] and [4] all require access to physical systems for collecting training data. However, we aim to perform multi-target prediction of performance and power without having access to physical systems.

We have two objectives for the work in this section; first, build an accurate performance and power prediction model for physical systems without accessing them. Second, achieve the objective of computer systems selection using the model. We proposed a novel model "Cross Performance and Power Prediction with Scaling" to achieve the first objective. Our model used only gem5 [32] simulated systems dataset in conjunction with the machine learning model to perform predictions for physical systems. Furthermore, building full systems in the gem5 simulator is arduous [39] [18], hence we explored the gem5 emulation mode for achieving speed. However, gem5 emulated systems have a larger design gap with physical systems reducing the accuracy of the predictions which we compensated by applying application-specific "Scaling Factor". In summary, the contributions of this work include: First, we collected hardware features of real-world computer systems, made modifications to the gem5 simulator source and built 475 systems in the gem5 simulator using emulation mode. We executed well-known benchmark applications from SD-VBS and MiBench with different computations and data access patterns on gem5 simulated systems to construct the performance and power dataset. Second, we executed the same benchmark application on selected physical systems to build the performance and power dataset for validation. Third, we trained a multivariate machine-learning model from the gem5 simulated systems' dataset to predict the performance and power simultaneously for physical systems. Fourth, we designed a mathematical model to derive application-specific "Scaling Factor" to improve the accuracy of predicted values compared to the physical systems' actual values. Finally, we demonstrated that accurate predictions from our model achieved the objective of computer systems selection without the need to access physical systems.

The remainder of the section is organized as follows: Section 5.2.2 describes our methodology and model. Section 5.2.3 describes applications selection for workload and procedure to collect performance and power consumption on simulated systems and physical systems for cross prediction. Section 5.2.4 articulates the results from our cross prediction model. Section 5.2.5 provides details about using our cross prediction model for computer system selection. Finally, section

5.2.6 has concluding remarks along with tasks that we plan to continue in future work.

5.2.2 Methodology and Model

In this section, we describe our methodology and the cross prediction model that we propose.

5.2.2.1 Methodology

We proposed that instead of using physical systems to collect performance and power for selection, we quickly built systems in cycle-accurate simulators [89] for accurate performance and power measurements. We then trained a machine learning model to predict the performance and power of physical systems. Gem5 [32] simulator is a well-accepted cycle-accurate simulator used by academicians and industry researchers. Furthermore, McPAT [93] in conjunction with gem5 provides accurate power consumption of application execution as shown in [9]. We can build simulated systems with a full-system mode in gem5 for very accurate performance estimates compared to the physical system as shown in [9]. However, setting up many full systems in gem5 is arduous and slow as compared to native execution indicated by [39] and [18]. Therefore, to speed up the construction of systems, we built simulated systems in gem5 using system-call emulation mode.

We collected nine hardware features cpu clock speed, cores, instruction set architecture (ISA), L1 size, L2 size, L3 size, memory type, memory access speed, memory size from simulated systems. We trained a decision tree machine learning algorithm from these hardware features, the performance and power dataset collected from gem5 and McPAT by executing applications on gem5 simulated systems and predicted the performance and power of physical systems. Nevertheless, due to the design differences in the emulation mode of gem5 simulated systems and physical systems, we observed large errors between predicted values from the machine learning model and the actuals from the physical system. To improve the accuracy of predictions, we designed a mathematical model to deter-

mine the factor called the "Scaling Factor" by which predicted values needed to be adjusted. There are two components to the scaling factor; dissimilarity between the design of gem5 simulated systems with emulation mode and physical systems, and the application's dependence on hardware features resulting in further variations in performance and power. We determined both factors to derive the application-specific scaling factor. We applied the scaling factor to the predicted values for accurate performance and power prediction of the physical system.

5.2.2.2 Cross Performance and Power Prediction with Scaling Model

We categorize performance and power prediction problem as multi-target (multivariate) regression problem [92] due to the correlation between performance (runtime) and power consumption. To address the multivariate regression problem, our model trains a single machine-learning model to predict both performance and power simultaneously. In a separate study conducted by us [42], we have established that tree-based machine learning algorithms such as decision tree [90] [6] provides the highest prediction accuracy for the performance prediction model. Therefore, we have used a decision tree algorithm from python-based machine learning library scikit-learn [65] for our model.

Our model "Cross Performance and Power Prediction with Scaling" has three phases as shown in figure 5.6; training phase, cross performance & power prediction phase, and model evaluation phase. For the training phase, we denoted simulated systems hardware features as Xs and actual runtime and power as $Ys \in \mathbb{R}^2$. Since memory-type and instruction-set-architecture (ISA) features have text data, we encoded these features using a dummy encoding technique to convert them from text data in Xs to real value in $Xs' \in \mathbb{R}^d$. The objective of the training phase is to learn a function $\Im(Xs') \approx Ys$, which we referred to as the "Learned Model". In the training phase, we trained a single decision tree regression algorithm per application from 80% of gem5 simulated systems' performance and power dataset generated in section 5.2.3.2 consisting of real-valued hardware features $Xs' \in \mathbb{R}^d$ and actual performance and power $Ys \in \mathbb{R}^2$. We predicted the performance and power for the remaining 20% simulated systems by providing only hardware fea-

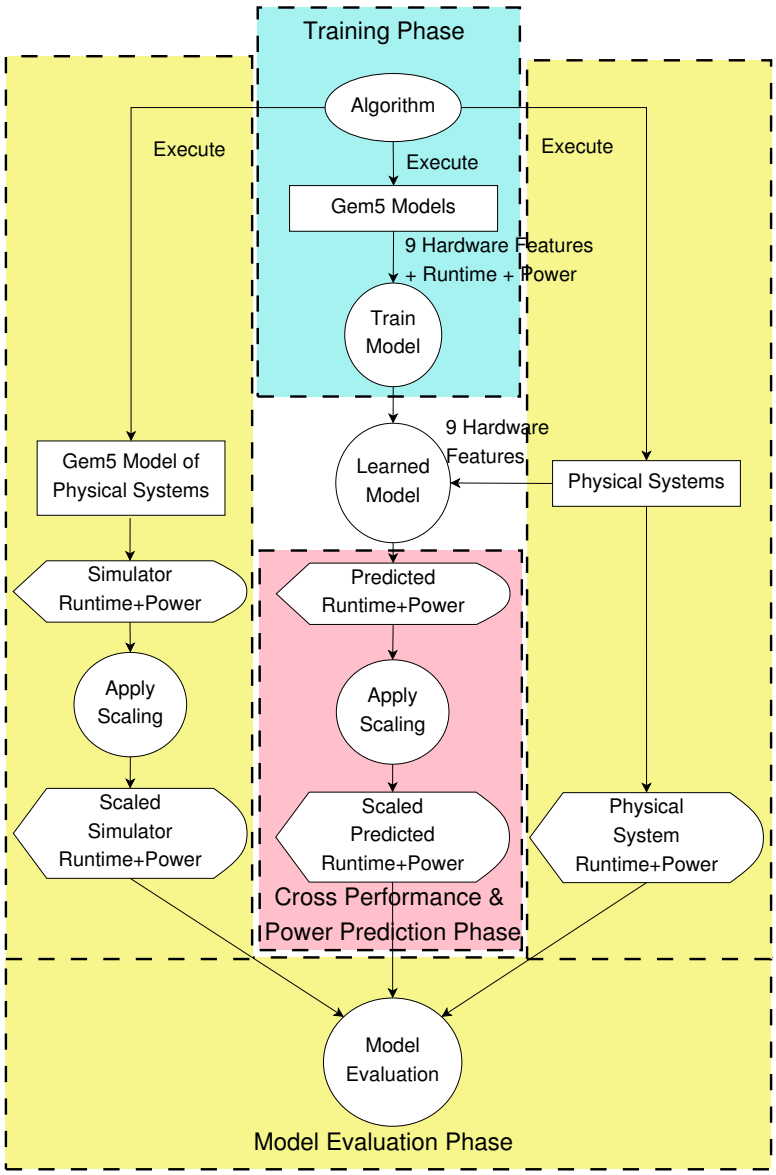


Figure 5.6: Cross Performance and Power Prediction with Scaling Model

tures to the "Learned Model" to validate the accuracy.

In the cross performance and power prediction phase; first, we collected nine hardware features Xp of each physical system using the dmidecode utility. We also collected the actual runtime and power consumption Yp as discussed in section 5.2.3.3. We encoded memory-type and ISA with text data from Xp to real-value in $Xp' \in \mathbb{R}^d$. We fed real-valued physical system features Xp' as an input to the application-specific "Learned Model" to predict the runtime and power $Yppred$ such that $\Im(Xp') \rightarrow Yppred$, which we labeled to as cross predicted values. However, the cross predicted values $Yppred$ when compared to physical systems' ac-

tual Yp results in large errors. We believe that these large errors are caused by machine learning models trained only from simulated systems' dataset to predict performance and power for physical systems, indicating dissimilarities between simulated and physical systems environment. To identify these dissimilarities and improve the accuracy of our model, we perform the model evaluation phase.

In the model evaluation phase, upon analyzing the differences between Yp and $Yppred$ for different benchmark applications, we observed that two factors are causing large errors. First, the design of gem5 simulated systems built using emulation mode differs from physical systems resulting in a major factor for errors. Second, variations in a major factor, called a minor factor, are caused by the dissimilarities in the dependence on hardware features by compute-intensive or data-intensive applications. The combined effect of major and minor factors is what we call the "Scaling Factor." As shown in figure 5.6, we applied the application-specific "Scaling Factor" to the cross predicted values to reduce the error between cross predicted values and the actual physical system values. We discuss the mathematical formulation of the "Scaling Factor" below:

5.2.2.3 Determining Scaling Factor

The scaling factor is determined by a major and minor factor as shown in equation 5.1.

$$ScalingFactor(SF) = MajorFactor + MajorFactor * MinorFactor \tag{5.1}$$

To calculate the scaling factor, we first determined a major factor, large errors caused by dissimilarities in gem5 simulated systems built-in emulation mode and physical systems. First, we constructed gem5 simulated systems using the same hardware features as the physical systems as shown in the left yellow lane in figure 5.6. We then executed the benchmark applications on these gem5 simulated systems equivalent to physical systems and collect the actual performance and power dataset Ysp. We took the mean difference between the Ysp and Yp for

performance and power separately, which is a major factor.

$$MajorFactor = \frac{\sum_{n=1}^{N}(Ysp_n - Yp_n)}{N} \tag{5.2}$$

To determine an application-specific minor factor, we first needed to quantify the application's dependence on the processor, cache and memory features resulting in variations in performance (runtime). We calculated the Pearson correlation (PC) coefficient between individual hardware features and simulated systems performance (runtime) for each application.

We define processor feature set as computational features (C) and memory features (M) as shown.

Computational Features(C)={numcpus, L1size, L2size, L3size}

Memory Features(M)={memclock, memsize, memtype*}

We calculated the correlation coefficient for computational features PC_C and memory features PC_M as shown in equations 5.3 and 5.4. The minor factor is calculated using PC_C and PC_M as shown equation 5.5.

$$PC_C = \frac{\sum_{c \in C} PC(c)}{|C|} \tag{5.3}$$

$$PC_M = \frac{\sum_{m \in M} PC(m)}{|M|} \tag{5.4}$$

$$MinorFactor = \frac{PC_C - PC_M}{PC_C} \tag{5.5}$$

5.2.3 Experimental Setup and Dataset Construction

In this section, we articulate the selection of benchmark applications, provide details of systems constructed in the gem5 simulator using emulation mode and the selection of physical systems used for the validation of our model. We also outline the procedure for collecting actual performance (runtime) and power consumption from gem5 simulated systems and physical systems for selected applications.

Table 5.3: Applications used as workloads for Cross Performance and Power Prediction Model with Scaling

Application	Benchmark	Intensiveness
mser	SD-VBS[70]	compute-intensive
svm	SD-VBS[70]	compute-intensive
tracking	SD-VBS[70]	data-intensive
stitch	SD-VBS[70]	compute-intensive and data-intensive
dijkstra	MiBench[63]	data-intensive
sha	MiBench[63]	compute-intensive

compute-intensive = compute-bound
data-intensive = memory-bound

5.2.3.1 Application Selection for Workload

We selected applications from different application domains having different computations and memory access patterns. Our selection includes four applications svm, tracking, stitch and mser from San Diego Vision Benchmark Suite (SD-VBS) [70] and two applications, sha and dijkstra, from MiBench [63], as shown in table 5.3. The svm is a machine learning algorithm, stitch and tracking are image operations, sha is an encryption algorithm, and dijkstra is a network protocol. According to SD-VBS documentation [70], mser and svm are compute-bound (compute-intensive) having a higher dependency on processor features, tracking is memory-bound (data-intensive) having a higher dependency on memory features, whereas stitch depends on both, computation and memory access (compute-plus-memory-bound). The correlation coefficient between systems' hardware features and runtime can identify features of processors and memory that dominate the runtime categorizing the respective application into compute-intensive or data-intensive respectively, as shown in [11].

For each application in table 5.3, we calculated Pearson correlation coefficient [94] (PC) between simulated systems runtime and computational features (C) and memory hardware features (M) as shown in columns B to K of table 5.4 with columns G to J showing the PC values for the encoded memory-type feature. We also calculated the application-specific PC_C, PC_M and minor factor using equations 5.3, 5.4 and 5.5 as shown in columns L, M and O. We observed from the table 5.4 that for mser and sha $PC_C \gg PC_M$ categorizing them as compute-bound

Table 5.4: Runtime Minor Factor Based on Pearson Correlation Coefficient

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
dijkstra	0.44	0.41	0.40	0.02	0.26	0.08	0.31	0.79	0.75	0.18	0.36	0.44	MB	-0.22
sha	0.21	0.29	0.20	0.02	0.06	0.16	0.20	0.12	0.16	0.34	0.26	0.14	CB	0.46
stitch	0.07	0.38	0.27	0.00	0.12	0.18	0.28	0.34	0.34	0.23	0.24	0.25	CB+MB	0.06
tracking	0.29	0.27	0.31	0.00	0.12	0.24	0.37	0.38	0.41	0.18	0.26	0.31	MB	-0.17
mser	0.05	0.61	0.20	0.00	0.03	0.14	0.20	0.13	0.11	0.33	0.30	0.12	CB	0.59
svm	0.38	0.31	0.23	0.00	0.07	0.35	0.32	0.13	0.21	0.14	0.26	0.22	CB	0.18

A-Algo, B-PC_{L1Size}, C-PC_{L2Size}, D-PC_{L3Size}, E-$PC_{NumCPUs}$, F-$PC_{MemSize}$, G-$PC_{MemType1}$, H-$PC_{MemType2}$, I-$PC_{MemType3}$, J-$PC_{MemType4}$, K-$PC_{MemClock}$, L-PC_C, M-PC_M, N-Algo Type (CB/MB), O-MinorFactor

applications whereas in case of tracking and dijkstra $PC_C \ll PC_M$ categorizing them as memory-bound. In the case of stitch, we observed that $PC_C \approx PC_M$ categorizing it as compute-plus-memory bound. In other words, a positive minor factor indicates compute-bound applications, a negative minor factor indicates memory-bound applications and a minor factor close to zero indicates the application has dependencies on both processor and memory features.

5.2.3.2 Simulated Systems for Training

We built 475 systems in the gem5 simulator with system-call emulation mode as discussed in chapter 2.3.1 using the features collected from real systems as shown in table 2.1. However, due to the limitations and differences in the design of gem5 simulated systems compared to physical systems as indicated in chapter 2.3.1, simulated systems are close but not the exact representation of physical systems.

We executed each benchmark application from table 5.3 on each of the 475 simulated systems to collect actual runtime along with hardware features. We utilized the McPAT [93] tool to collect the power consumption for each application execution on gem5 simulated systems as carried out in our earlier work [45].

5.2.3.3 Physical Computer Systems for Cross Performance and Power Prediction

For cross performance and power prediction experiments, we kept our scope limited to x86-based computer systems mainly using Intel processors. We consid-

Table 5.5: Physical Computer Systems for Cross Performance and Prediction Model with Scaling

Sr	ISA	CPU Speed GHz	Cores	Mem Type	Mem Access MHz	Mem Size GB	L1-L3 Cache Size**
A	x86	3.2	4	DDR3	1600	4	32,6
B	x86	3	2(4)	DDR3	1600	8	32,4
C	x86	3.2	6(12)	DDR4	2666	16	32,12
D	x86	3.2	4	DDR3	1600	4	32,6

**L1 Cache Size is in kB and L3 Cache Size is in MB
Configuration taken from following models
A.Intel Core i5-6500 B.Intel Core i7-6500U C.Intel Core i7-8700 D.Intel Core i5-3470

ered physical systems with different feature values of all nine hardware features, as shown in table 5.5. The selection of these systems consists of a server system (C), two general-purpose desktop systems (A) and (D) and a laptop (B). We chose these systems to consider different power profiles, with the server system having the highest power consumption and a laptop having a mobile processor with the lowest power consumption. Furthermore, we chose systems B and C which use hyper-threading with 2 threads per core, as shown in the Cores column.

We first collected hardware feature values from each of the systems using the dmidecode utility. We modified each of the benchmark applications' source code to integrate PAPI-API [95] to collect power consumption. PAPI-API uses Intel's RAPL [96] to read machine status registers (MSRs) to collect power consumption. We executed modified versions of each benchmark application on each physical system to collect runtime and power consumption from benchmark logs.

5.2.4 Results

In this section, we perform validation in three ways. First, we validated the prediction accuracy of the learned model, a decision tree machine learning algorithm trained only from the performance and power dataset collected from simulated systems. Second, to evaluate the accuracy of the cross prediction model, we compared cross predicted runtime and power $Ypred$ to the actual runtime and power Ysp gathered from gem5 simulated systems having hardware features same as

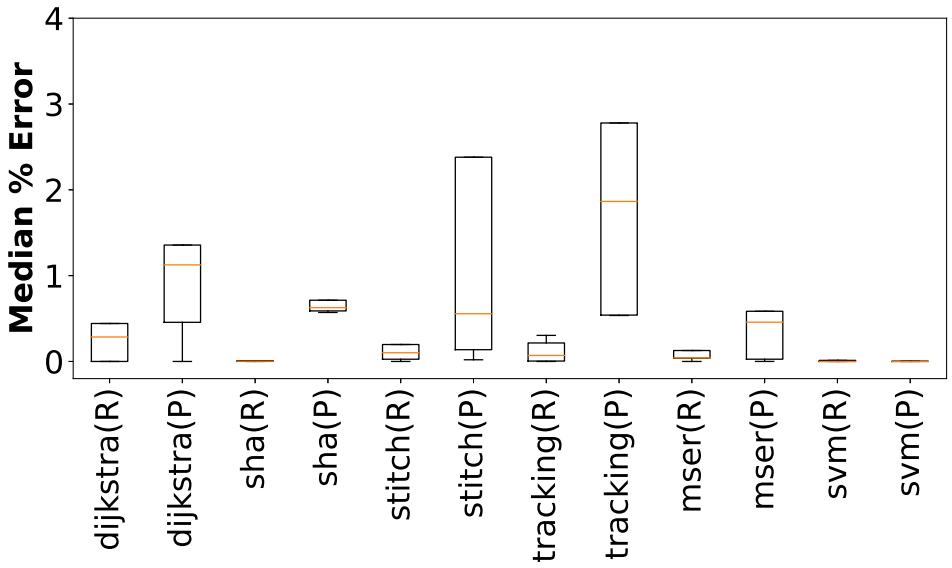


Figure 5.7: Cross-Validation Mean Percentage Error (R)-Runtime, (P)-Power

physical systems. Third, to evaluate the accuracy of our model with scaling factor, we compared the cross predicted runtime and power $Yppred$, before and after applying the scaling factor, to the actual physical systems' runtime and power Yp.

5.2.4.1 Simulation-based Performance Prediction Accuracy

To evaluate the accuracy of our decision tree algorithm, we performed 5-fold cross-validation on simulated systems' performance and power dataset [Xs, Ys]. For each fold, we trained a new decision tree algorithm on 80% of randomly selected samples from [Xs, Ys] and the remaining 20% from [Xs] used for prediction. The median percentage errors of all five folds for all applications are shown in figure 5.7 with a maximum error of 2.1%, achieving an accuracy of about 98%.

5.2.4.2 Cross Performance and Power Prediction with Scaling Model Accuracy

To demonstrate the accuracy of our model, we performed two tests. In the first test, we compared unscaled cross predicted runtime and power $Yppred$ with Ysp, the actual runtime and power from gem5 simulated systems modeled using features of physical systems. In the second test, we compared the actual runtime and power from physical computer systems Yp to unscaled and scaled cross predicted runtime and power $Yppred$.

5.2.4.2.1 Cross Predicted Targets (Yppred) vs Actual Targets from Gem5-based Physical Systems (Ysp)

We plotted unscaled cross predicted values $Yppred$ and the actual performance and power from simulated systems with identical features as physical systems Ysp in figures 5.8 and 5.9. We make two observations from these results. First, regardless of the application-type, compute-bound or memory-bound, runtime predictions have higher prediction accuracy than power predictions ($|Yppred[runtime] - Ysp[runtime]| < |Yppred[power] - Ysp[power]|$) due to smaller variations in runtime as compared to the variations in power. We standardized the runtime and power values for comparison and plotted variations between standardized runtime and power values with respect to its respective mean for tracking application as an example in figure 5.10. We can observe that most of the standardized runtime fits in the window of -0.1 to 0.7 with a mean of 0.35 whereas standardized power ranges from -0.1 to 1 with a mean of 0.42. We make a similar observation for other applications as well.

The second observation is that the difference in power values of Ys and $Yppred$ for compute-bound applications is lower than memory-bound applications. We plotted the histogram in figure 5.11, showing the spread of power values between compute-bound mser versus memory-bound tracking. For the tracking application, the histogram has power values distributed from four to twelve whereas for mser they are from four to ten. The broader spread in tracking is because gem5 has an implementation of several memory devices, resulting in higher variations for memory-bound applications. On the other hand, gem5 supports only one out-of-order O3CPU processor which is used in constructing all simulated systems causing lower variations in compute-bound applications.

5.2.4.2.2 Cross Predicted Targets (Yppred) vs Actual Targets from Physical Systems (Yp)

For the second test, we compared cross predicted values of runtime and power to the actual runtime and power of physical systems as shown in figures 5.12 and 5.13. The red bar indicates the actual runtime and power of physical systems.

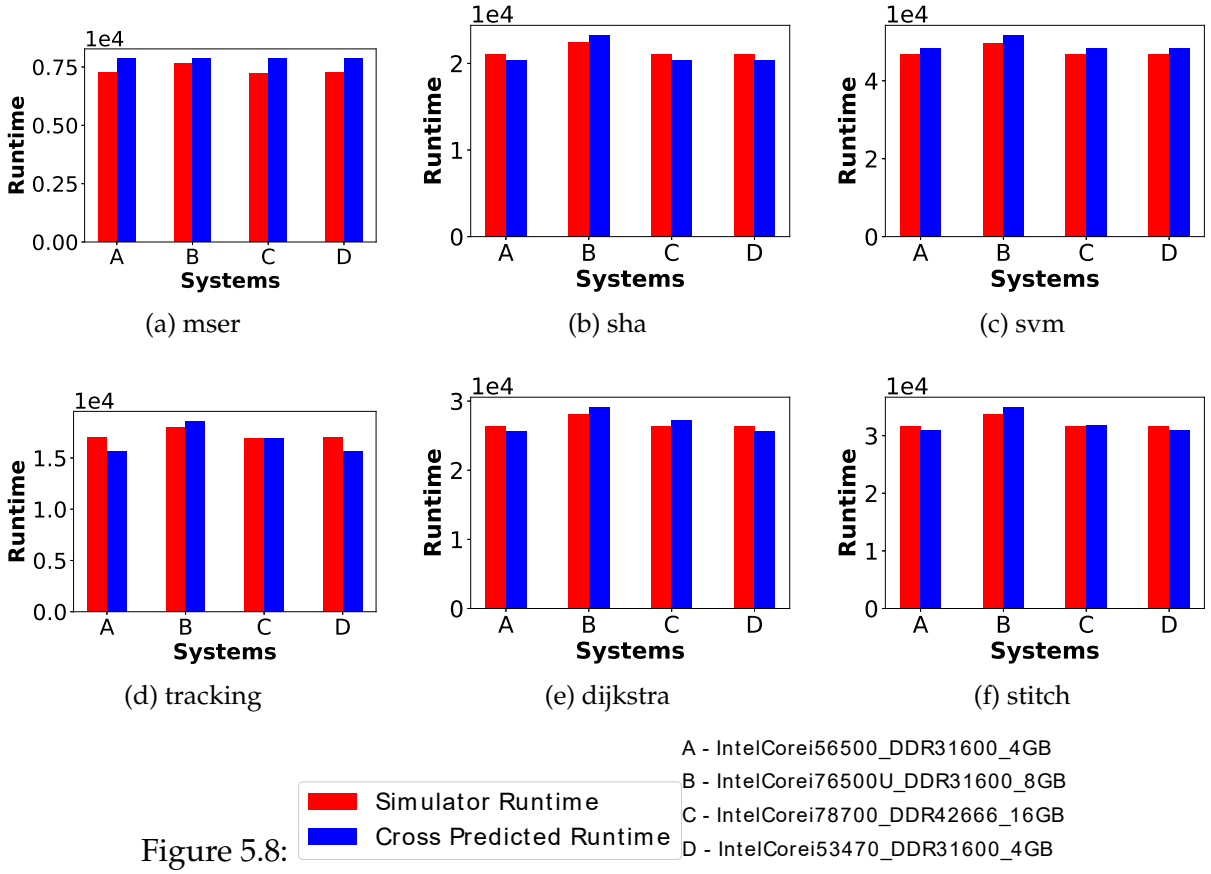


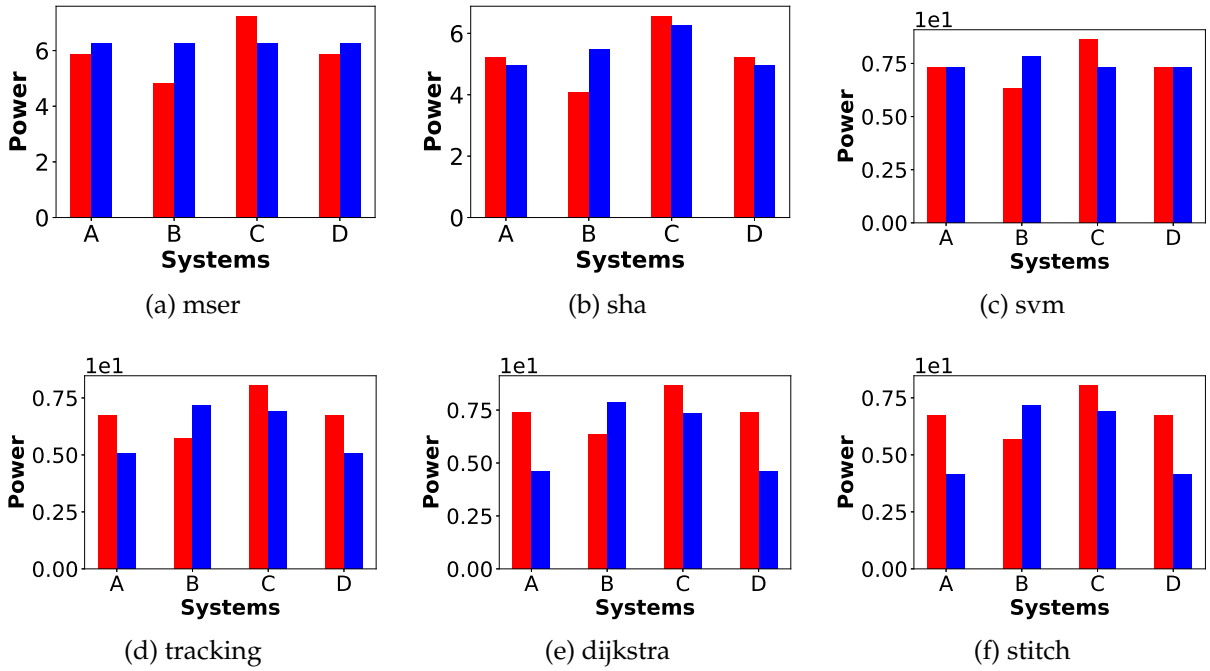
Figure 5.8: Simulator Runtime (μs) vs Cross Predicted Runtime (μs)

The black bar depicts the unscaled cross predicted values (i.e. before applying the scaling factor) whereas the blue bar shows scaled cross predicted values (i.e. after applying the scaling factor). Thus, we expect to have a red bar (actual runtime or power) height much closer to the height of the blue bar representing scaled cross predicted value rather than the height of the black bar representing unscaled cross predicted value.

We make following two observations from figure 5.12, which compares cross predicted unscaled and scaled runtime $Yppred[runtime]$ to the physical systems' runtime $Yp[runtime]$:

Observation 1: Physical system D has a similar configuration as system A, however, it has much higher runtime values causing it to have higher prediction errors especially for tracking, dijkstra, stitch, and svm applications. This is because system D has a much lower bus speed than system A.

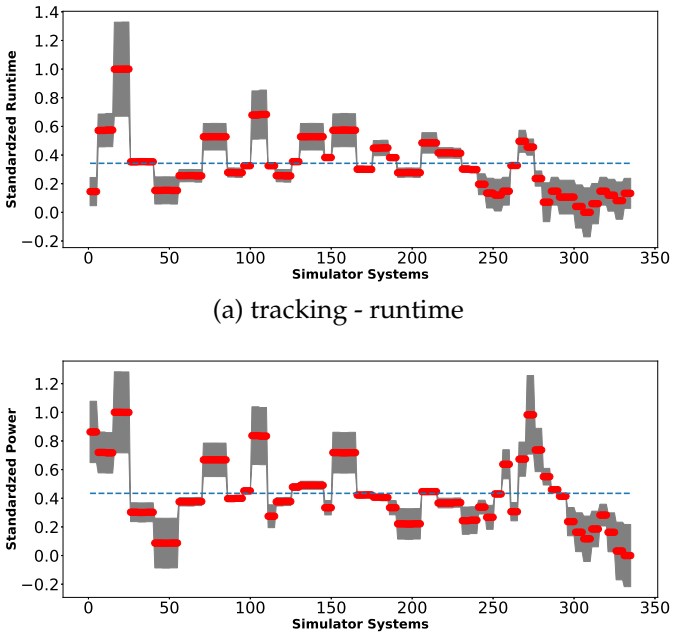
Observation 2: Difference (error) between actual runtime from the physical



(a) mser (b) sha (c) svm

(d) tracking (e) dijkstra (f) stitch

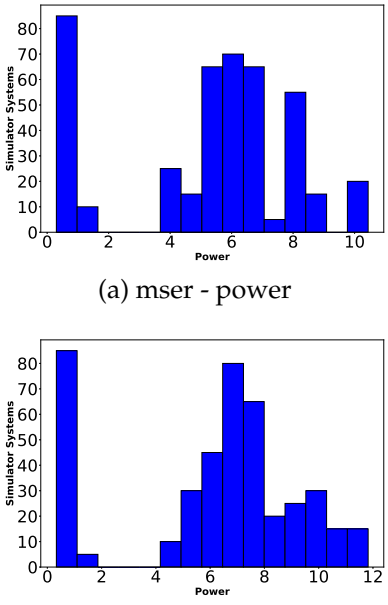
Figure 5.9: Simulator Power (W) vs Cross Predicted Power (W)



(a) tracking - runtime

(b) tracking - power

Figure 5.10: Variations in Runtime and Power With Respect To Mean



(a) mser - power

(b) tracking - power

Figure 5.11: mser (compute-bound) vs tracking (memory-bound) power histogram

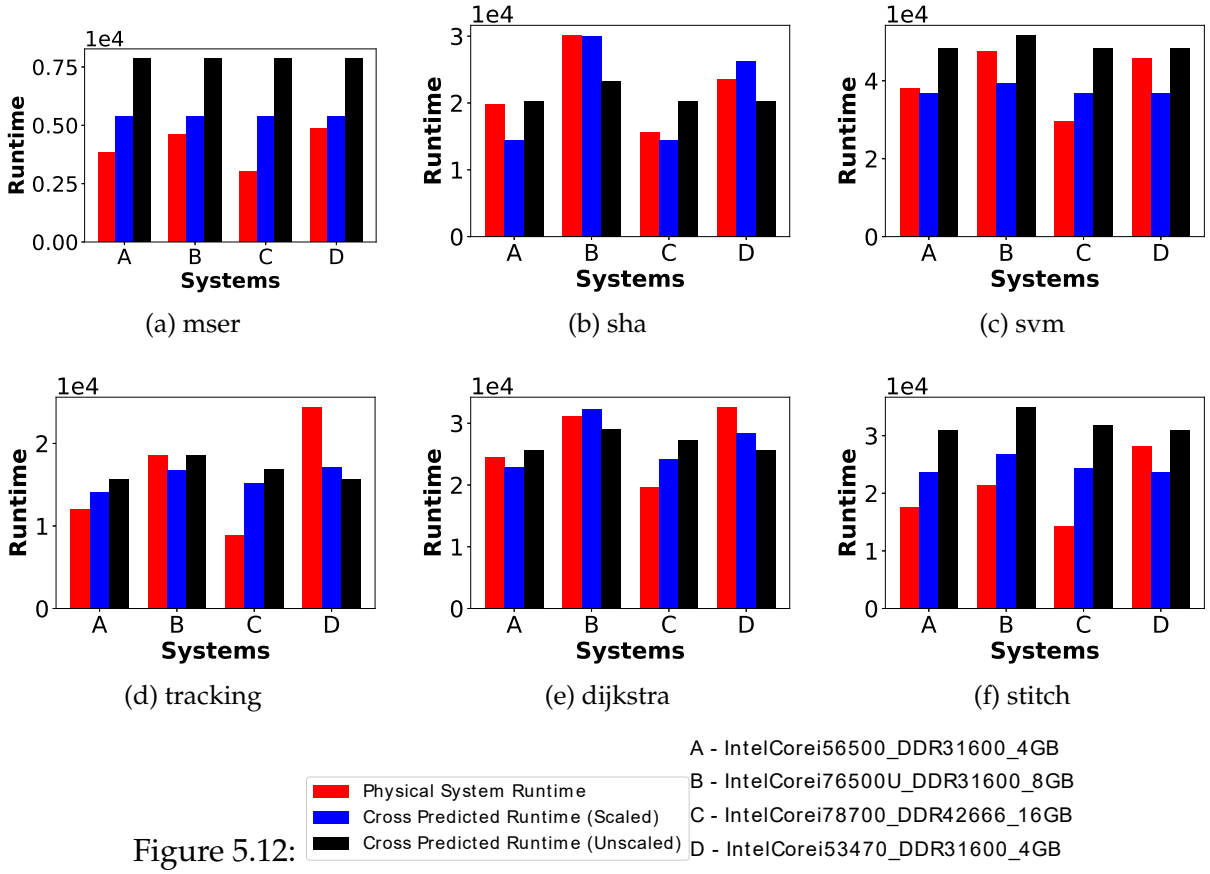


Figure 5.12: Physical System Runtime (μs) vs Cross Predicted Runtime (μs)

system $Yp[runtime]$ and the unscaled cross predicted runtime $Yppred[runtime]$ is larger in compute-bound applications as compared to memory-bound applications. For example, mser and sha have 48% and 18% errors respective with a mean of 30% whereas dikstra has an error of 12%. When comparing two compute-bound applications, mser has a much higher error compared to sha. This is because mser has higher dependence on processor features than sha as per columns L and M in table 5.4.

The higher dependence of application on processor features increases the gap in runtime due to greater dissimilarities in processors than memory devices between physical systems and gem5 simulated systems. The gem5 simulated systems were built with the only supported superscalar processor witnesses a significant gap in features with that of several disparate processors used in physical systems resulting in higher runtime difference for compute-bound applications causing the higher error. On the other hand, the support of several different

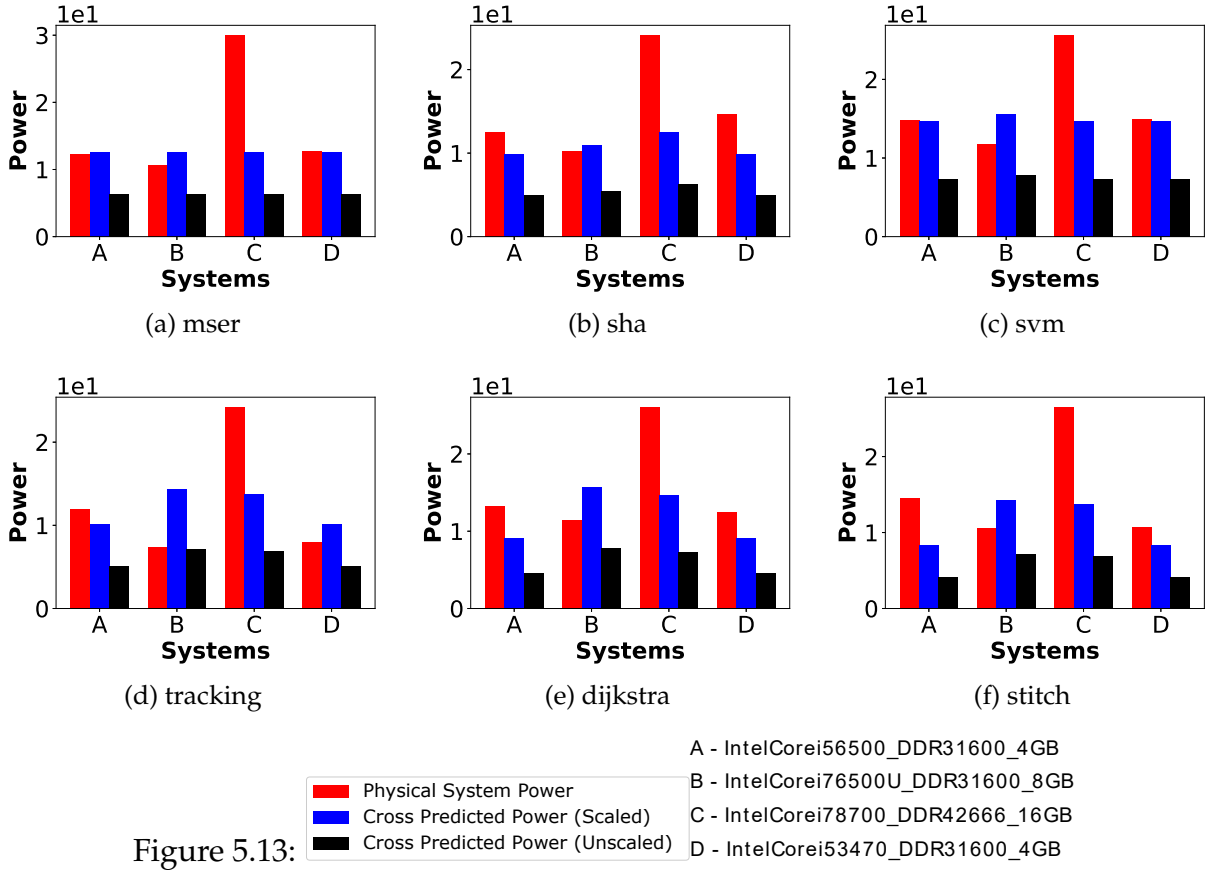


Figure 5.13: Physical System Power (W) vs Cross Predicted Power (W)

memory devices in gem5 enables the simulated systems' memory to closely represent physical systems' memory resulting in a small error in runtime for memory-bound applications. For memory-bound applications also, differences in processors of simulated systems and physical systems do exist but it has less impact due to higher dependence on memory which is much slower than a processor.

To calculate the scaling factor, we first calculated the major factor. The mean percentage error for compute-bound and memory-bound applications between physical systems and gem5 simulated systems built using identical features as physical systems is about 30% and 10% respectively with a mean of 20%. Hence, we considered the value of 20 as a major factor. The minor factor is already calculated using the Pearson correlation coefficient as shown in column O of table 5.4. Using the formula in equation 5.1, we calculated the scaling factor for each benchmark application which is 15.63 for dijkstra, 29.24 for sha, 18.72 for stitch, 16.58 for tracking, 31.83 for mser and 23.56 for svm. The application-specific scaling factor

is then applied to unscaled cross predicted runtime to calculate the scaled cross predicted runtime.

Both unscaled and scaled cross predicted runtime values are shown in figure 5.12. We can confirm from the plot that scaled runtime errors are much lower as compared to unscaled runtime errors when unscaled and scaled cross predicted runtimes are compared with the physical system's actual runtime. For each application, scaled cross predicted runtime mean percentage error of 10% to 25% is achieved considering all physical systems combined.

We have following observations from figure 5.13 which compares unscaled and scaled cross predicted power $Yppred[power]$ and physical systems power $Yp[power]$:

Observation 3: Server system C consumes maximum power compared to other systems due to a more number of cores and large memory, causing it to have the highest difference between cross predicted power $Yspred[power]$ and the actual physical system's power $Yp[power]$. System B having a mobile processor consumes marginally lower power than all other systems resulting in higher errors than systems A and D.

Observation 4: For both types of applications, compute-bound or memory-bound, the mean error between the unscaled cross predicted power $Yspred[power]$) and physical system's power ($Yp[power]$) is approximately equal. This is because power consumption from the memory device is insignificant compared to a processor. For example, for compute-bound mser, the total power consumption of 29.97 Watts is divided into package power consumption (computation power) of 29.36 Watts and memory power of 0.61 Watts, whereas for memory-bound dijkstra package power consumption of 25.38 Watts and memory power of 0.66 Watts with the total power consumption of 26.04 Watts. Therefore, there is a negligible difference in the scaling factor for power consumption of compute-bound versus memory-bound applications. Furthermore, we observed errors of about 100% in all physical systems with the exclusion of server system C.

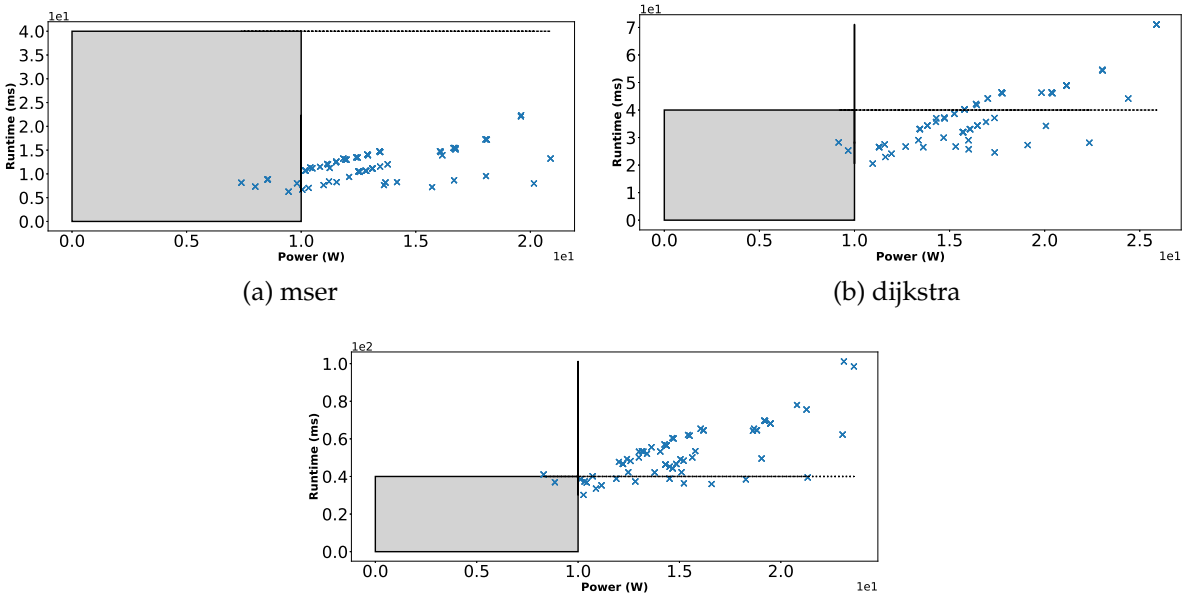
Considering observation 4, we applied the power scaling factor of 100% uniformly for all unscaled cross predicted power values. Figure 5.13 shows both

unscaled and scaled cross predicted power values. From the plot, we can infer that differences between scaled power and the physical system's power are much lower as compared to the difference between unscaled power and the physical system's power. We reduced errors for each application in the range of 6% to 40% for all systems with a mean of 25% excluding the server system C. A different scaling factor would apply to server systems as it represents a different class of computer systems.

5.2.5 Selection of Computer Systems

Performance and power consumption are essential factors in the selection of computer systems today. For example, in real-time applications, performance (runtime) is more important; therefore, systems with lower runtime are preferred with compromise in power consumption. On the other hand, systems used in the area such as combat require to run with limited power; therefore, systems with a small power footprint are preferred over performance. In this section, we demonstrate that our proposed cross performance and power prediction model with scaling can effectively be used to select physical systems based on user budget of runtime or power or both values without access to the physical systems.

We have already established that our cross prediction model predicts both performance and power with high accuracy from the simulated systems. Figure 5.14 displays performance (runtime) versus power values after applying the respective scaling factor for mser, a compute-bound application, dijkstra, a memory-bound application, and stitch, compute-plus-memory-bound application. Let us assume that the user provides a runtime budget of 40 milliseconds or a power budget of 10 Watts or both. Thus, systems that take maximum runtime of 40 milliseconds and/or require a maximum of 10 Watts of power must be selected. From the figure 5.14, any systems to the left of the vertical line can be selected for power consumption of 10 Watts and lower, while any of the systems below the horizontal line can be selected for the runtime of 40 milliseconds and lower. The systems in the shaded region fulfill both runtime and power budget.



(a) mser

(b) dijkstra

(c) stitch

Figure 5.14: System Runtime vs Power

5.2.6 Summary

In this section, we proposed a novel model "Cross Performance and Power Prediction with Scaling". We established that our model can accurately predict the performance and power of various applications for physical systems from decision tree machine learning models trained only on performance and power datasets collected from gem5 simulated systems built using emulation mode. Prediction accuracy of six applications from MiBench and SD-VBS benchmarks has achieved a prediction error of 10% to 25% for performance and 6% to 40% for power for general-purpose systems. The benefit of our model is that it predicts the performance and power of physical systems without the need to access them for executing the applications. Accurate predictions from our model are the result of the systematic derivation of application-specific "Scaling Factor", which when applied to cross predicted values improves the accuracy significantly. We have demonstrated that users can utilize our model to select physical computer systems given a budget of runtime and/or power. Currently, our study has been limited to the x86-based systems; therefore, we plan to extend our work by considering systems with other instruction sets and networked systems in the future.

CHAPTER 6

Cross Prediction with Transfer Learning using Machine Learning

6.1 Cross-Platform Performance Prediction with Transfer Learning using Machine Learning

6.1.1 Overview

Advancements in computer systems have helped us with the availability of several options of computer systems each having diverse hardware features. An application when executed on these systems results in dissimilar performances (runtimes). Therefore, it is essential to select system(s) that provide optimal performance for an application. It is possible that access to some physical systems is available but access is not available for the systems that provide optimal performance for an application. Thus, we need to predict the performance of an application for the unavailable physical systems from the performance data of available physical systems.

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 available target platform data (that we have access to) to predict the performance data of the unknown (unavailable) target platform. In this work, we explored transfer learning to solve two prob-

lems; cross-systems prediction and cross-platform prediction. In cross-systems prediction, we predicted the performance of physical systems from performance collected from simulated systems while predictions of the dissimilar target architecture (such as x86 and ARM) are performed in cross-platform prediction.

The remainder of the section is organized as follows: Section 6.1.2 describes work related to transfer learning and how our objective differs. Section 6.1.3 describes the selection of benchmarks for workload and computer system selection for simulated systems and physical systems. Section 6.1.4 details machine learning models along with techniques to improve prediction accuracy, such as grid search that we have used for the study. Section 6.1.5 provides results and analysis of cross-platform and cross-systems prediction, and evaluation of various machine-learning models including dimensionality reduction, hyper-parameter selection. Finally, section 6.1.6 provides concluding remarks and work that we plan to continue.

6.1.2 Related Work

Transfer learning has been widely used in performance prediction research. Work in [23] predicts 90% of the performance data of server C using machine learning model trained on 100% features collected from servers A and B plus only 10% from server C. [5] shows that a trained machine-learning model from one x86-based HPC system's performance data can be re-trained using transfer learning only on one percent of samples (data) 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 [25] for measuring the performance of MariaDB database's query responses installed on virtual machines when systems configurations and database sizes with incremental changes in system configuration or database size. Service-level metric predictions of data-center 2 or 3 are performed using transfer learning from measurements collected from data-center 1 in [24]. Cross-platform prediction for ARM-based system from an x86-based system is performed in [50].

The work in [23], [24] and [5] uses transfer learning for the prediction of tar-

get systems having the same instruction-set (x86) as the systems used for training while [50] performs cross-platform prediction without transfer learning. In addition, all these research works use physical systems to collect the data for training as well as prediction. To the best of our knowledge predicting the performance of physical systems from the simulated systems (cross-systems) utilizing the transfer learning approach is not addressed. Therefore, we address these four questions out of which the first two are addressed using transfer learning: (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 performance data? (iii) What is the effect of dimensionality reduction? (iv) How to select hyper-parameter values of machine-learning models?

6.1.3 Dataset Preparation

6.1.3.1 Applications Selection for Workload

We evaluated models described in section 6.1.4 on several benchmark applications representative of real-world applications having different computation and data access patterns listed in table 6.1. We selected compute-bound (CB) applications svm from SD-VBS [70], miniFE from the Mantevo benchmark suite [59], sha from MiBench and EP application from the NAS parallel benchmark (NPB) [58]. Similarly, selected memory-bound (MB) applications are dijkstra from MiBench and MG applications from NPB. Similarly, we selected quicksort from MiBench, and stitch from SD-VBS are dependent on both computation and data access making them compute-plus-memory-bound (CB+MB). These applications were executed on the simulated systems and physical systems from which the runtimes were collected.

6.1.3.2 Computer Systems Selection

We selected computer systems that are the class of computers commonly used today to execute the benchmark applications for collecting performance data. For

Table 6.1: Application and System Types used for Cross Performance Prediction Model with Transfer Learning

Applications	Benchmark	System Type	(MB/CB)	Data Points
dijkstra	MiBench	Physical	MB	52
dijkstra	MiBench	Simulated	MB	475
quicksort	MiBench	Physical	CB+MB	320
quicksort	MiBench	Simulated	CB+MB	2850
sha	MiBench	Physical	CB	52
sha	MiBench	Simulated	CB	475
miniFE	Mantevo	Physical	CB	124
EP	NPB	Physical	CB	108
MG	NPB	Physical	MB	108
stitch	SD-VBS	Physical	CB+MB	52
stitch	SD-VBS	Simulated	CB+MB	475
svm	SD-VBS	Physical	CB	52
svm	SD-VBS	Simulated	CB	475

CB = Compute-Bound (compute-intensive)
MB = Memory-Bound (data-intensive)

constructing performance dataset for physical systems, we chose 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 6.2. We collected the hardware features for these systems using the dmidecode utility. We executed each application with a number of processes from one to two times the number of cores in the system; that is, we executed 24 processes on a system with 12 cores. This is to take the advantage of systems with hyper-threading. We extracted runtimes for each execution from the benchmark application logs.

For the simulated systems' dataset, we built 475 simulated systems using the gem5 simulator, a widely accepted simulator for architectural research as discussed in chapter 2.3.1. The gem5 simulated systems were built considering nine system hardware features, as shown in table 2.1 with values from the real memory and processor models commonly used in today's computers. We collected runtimes from gem5 logs after executing each application on each of the 475 simulated systems. To apply transfer learning, for cross-platform prediction in simulated system dataset, we utilized 120 simulated systems based on the ARM in-

struction set (ISA) as target platform and 355 simulated systems based on the x86 ISA as known (source) platform.

Table 6.2: Physical Computer Systems for Cross Performance Prediction Model with Transfer Learning

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
Configurations were taken from the 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

6.1.4 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 the output during the testing (prediction) phase. Here, X_i can be a multidimensional tuple. In our case, X_i is the feature vector corresponding to processor, cache and memory features of i^{th} system as listed in tables 6.2 and 2.1. And Y_i is the runtime or performance of an application collected from the i^{th} system with the hardware feature vector X_i.

6.1.4.1 Machine Learning Models

Machine learning models that perform prediction for real-value output such as performance (runtime) are called regression models. Regression machine learning models are built utilizing various algorithms. The diverse classes to which

the algorithms belong to are linear, bayesian, tree-based, gaussian, support vector, nearest neighbor, ensemble and neural network models. The most desirable regression algorithm models providing accurate predictions depend on the dataset features like dimensionality, variance, skewness, sparsity, and multivariate probability distribution. The model algorithms that we utilized for the evaluation are listed in table 6.3. These model algorithms incorporate almost all the classes that we have listed above. Furthermore, our neural network model is a deep neural network (DNN) model with the summary, as shown in figure 6.1. For DNN, we used loss function as mean absolute error, optimizer as adam [81], and ran the model for 100 epochs. ReLU activation is used for the first four layers and linear activation for the last layer.

Table 6.3: Machine Learning Models for Cross Performance Prediction Model with Transfer Learning

Models Used	Abbr.	Important Hyper-Parameters
Support Vector Regr.[71]	svr	C, gamma, kernel
Linear Regr.	lr	fit_intercept, normalize
Ridge Regr.[72]	rr	alpha, fit_intercept, normalize, solver
K-Nearest Neighbor Regr.[73]	knn	p, n_neighbors, weights
Gaussian Process Regr.[74]	gpr	alpha, normalize_y, optimizer
Decision Tree Regr.[97]	dt	criterion, max_depth, max_features
Random Forest Regr.[75]	rf	n_estimators, criterion, max_depth
Extra Trees Regr.[76]	etr	n_estimators, criterion, max_depth
Gradient Boosting Regr.[77]	gbr	n_estimators, criterion, max_depth, loss
eXtreme Gradient Boosting[78]	xgb	n_estimators, max_depth, learning_rate
Deep Neural Network[98]	dnn	activation, loss, optimizer, metrics

6.1.4.2 Grid Search

Grid search technique [65] is used to fine-tune a model for the best performance on a specific dataset using cross-validation over a given parameter grid. Table 6.3 lists the model algorithms with their respective hyper-parameters that were tuned for performing each experiment. The essential hyper-parameters which had a significant impact on model prediction accuracy were used for the grid search.

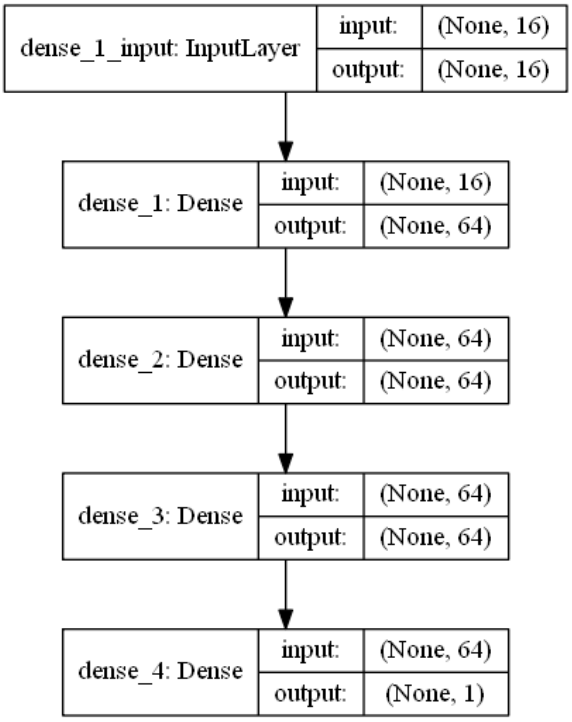


Figure 6.1: DNN Model Summary

6.1.4.3 Transfer Learning

Transfer learning aims to reuse previously gained knowledge from a specific task to apply the same to a similar but different task. It is tedious to build separate models for every new prediction problem. To save time and computational cost, one can build a generalized model from one problem that can be reused after fine-tuning on other similar problems. For example, Alex-net architecture [99], is a generalized model for image classification and, after little fine-tuning, can be used for similar image classification tasks.

6.1.5 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. We normalized the dataset using $StandardScaler()$ function from the scikit-learn library [65]. Furthermore, $ShuffleSplit()$ function [65] from scikit-learn library is used for cross-validation

to ensure the robustness of the models. Please note that we normalized both input and output data points, for cross prediction whereas we just normalized the input for other experiments.

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 R^2 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 R^2 score of 1 means the model fits the data well. The negative R^2 value reveals that the model fit is even worse than the horizontal hyperplane.

We evaluated dataset characteristics and model accuracy in different contexts and scenarios. Firstly, the outcome from principal component analysis (PCA) and optimization of hyper-parameters through 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 with a different machine learning algorithm, and lastly, the comparison, according to various application types, is also conducted.

6.1.5.1 Effect of Dimensionality Reduction using PCA

Principle component analysis (PCA) is a statistical method that identifies the correlated variables (features) to transform a large number of input variables into uncorrelated variables with reduction. We applied PCA to observe the effect of the dimensionality reduction on the accuracy of performance prediction models. Figure 6.2a illustrates the effect of PCA in improving performance prediction accuracy. The experiments are performed on the quicksort application's simulated systems dataset. As can be seen, models have lower error with mean R^2 = 0.75 and mean MedAPE = 22.59% when PCA is applied in comparison to mean R^2 = 0.66 and mean MedAPE = 27.77% without PCA. The results are averaged over all models (svr, knn, gpr, etr) as shown in figure 6.2a. 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. A similar increase in model performance was also observed for other datasets when PCA is applied.

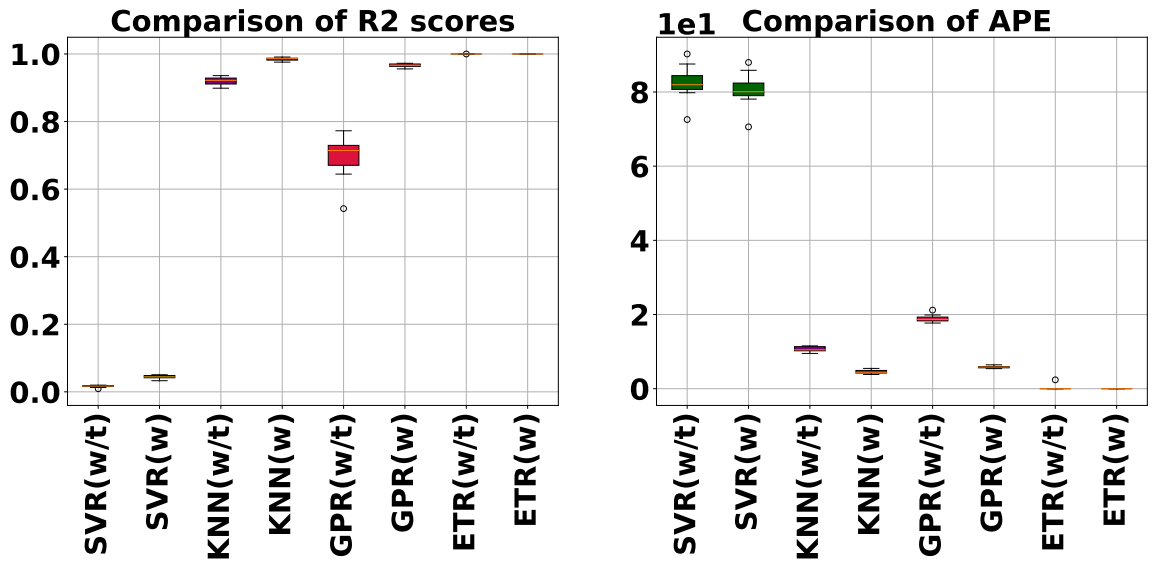
6.1.5.2 Effect of Grid Search on Model Performance

The 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-parameter values that provide the highest prediction accuracy from the model. Figure 6.2b illustrates the effect of using grid search for improving the performance of machine learning models. The experiments are performed on dijkstra application's simulated systems dataset. As can be seen, models have improved performance with mean R^2 = 0.69 and mean MedAPE = 16.66% when grid search is used in comparison to mean R^2 = 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 figure 6.2b. The improvement in prediction accuracy using grid search is due to the selection of suitable values of hyper-parameters 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 6.3 using an exhaustive grid search. In some cases, default values of hyper-parameters were the best selection as observed in the case of linear regression (lr) the figure 6.2b.

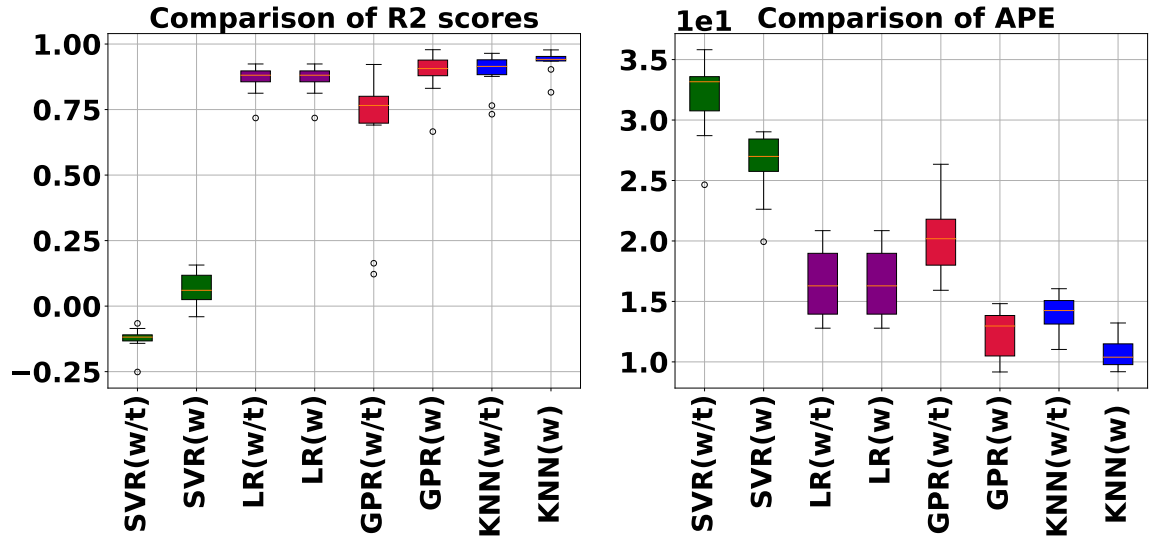
6.1.5.3 Cross Prediction using Transfer Learning

6.1.5.3.1 Cross-Systems Prediction: Simulated Systems to Physical Systems

To predict the runtime for physical systems from the simulated system's runtime, we implemented transfer learning on dijkstra (memory-bound) and sha (compute-bound) applications datasets to consider both types of applications (refer table 6.1 for dataset description). Firstly, for each application, we trained



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

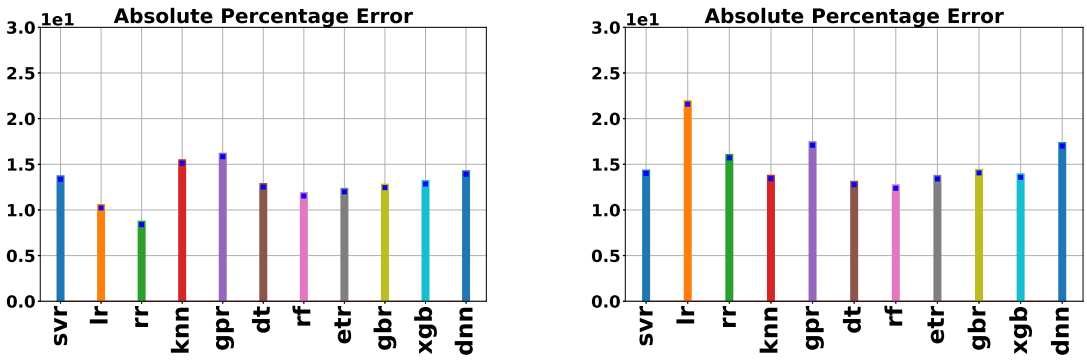


(b) Models with (w) and without (w/o) Grid Search on Dijkstra

Figure 6.2: Application of PCA and Grid Search Techniques

our models on 100% of simulated systems dataset and 10% of physical systems dataset. Since the partial_fit() function for retraining is not available for all algorithms in the scikit-learn library, we used the fit() function to train the model only once with a combined dataset of 100% of simulated systems and 10% of physical systems. After training the machine learning model, we fed hardware features as an input to the trained model from the remaining 90% of the physical systems dataset and predicted the performance. For implementing transfer learning in dnn, we first trained the model (shown in figure 6.1) on 100% of simulated

systems data. We then 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 a fine-tuned model on the rest 90% of physical systems data. The results from the transfer learning models for cross-systems prediction are shown in figure 6.3a for dijkstra and figure 6.3b for sha application. The first observation is among the two application types, where, memory-bound dijkstra with mean APE = 12.56% has 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 applications resulting in higher errors. Furthermore, we observed that linear regression models (lr and rr), performed best in the case of memory-bound applications (as shown in figure 6.3a), whereas bagging models (rf and etr) performed best in the case of compute-bound applications (refer figure 6.3b). Thus linear and bagging models are good for cross-systems performance prediction.



(a) dijkstra

(b) sha

Figure 6.3: Results for Training on Simulated System's Data and Testing on Physical System's Data

6.1.5.3.2 Cross-Platforms Prediction: x86-based Systems to ARM-based Systems

In this case, we used transfer learning to predict the runtime on ARM-based systems from x86-based systems on quicksort application's simulated systems dataset since the data points are maximum in simulated systems (refer table 6.1). We divided the dataset into two types of system instruction set architectures,

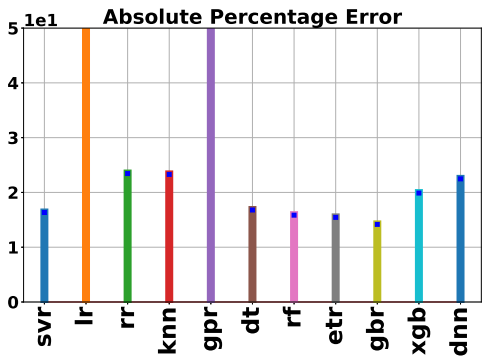
namely x86 and ARM. Firstly, we trained our models on 100% data points of x86-based systems (1470 data points) and 1% of data points of ARM-based systems (7 data points). After that, we predicted the performance of machine learning models on the rest 99% (713 data points) of ARM-based systems. Also, we performed transfer learning with dnn by fine-tuning the dnn model using 1% of test data in the same manner as explained in the above section. The results for the transfer learning models for cross-platform predictions are shown in figure 6.4a. 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 the case of simulated systems.

6.1.5.3.3 Cross-Platform Prediction: Intel Core to Intel Xeon

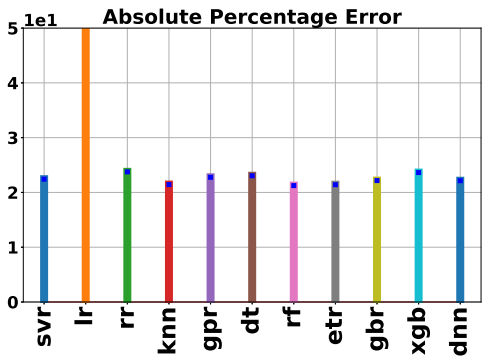
In this part, we used transfer learning to predict Intel-Xeon server systems runtime from Intel Core general-purpose systems runtime on quicksort application's physical systems dataset (refer table 6.1). We divided the dataset into two variants of the system's architectures, namely Intel Core and Intel-Xeon. Firstly, we trained our models on 100% data points of Intel Core-based systems (280 data points) and 10% of data points of Intel-Xeon systems (4 data points). After that, we predicted the performance of machine learning models on the rest 90% (36 data points) of Intel-Xeon systems. Also, we performed transfer learning with dnn in the same manner as explained in the above section. The results for the transfer learning models for cross-platform predictions are shown in figure 6.4b. We observed that tree-based models with a 22.30% mean percentage error have higher prediction accuracy for cross-platform performance prediction from Intel Core to Intel-Xeon architectures and outperform all other models including deep neural networks in the case of physical systems.

6.1.5.4 Model Comparison for Performance Prediction

Figure 6.5 depicts the values of R^2 score, MedAPE and, MSE for each model averaged over all datasets listed in table 6.1. We observed that according to the



(a) quicksort-simulated-x86 to ARM



(b) quicksort-physical-Intel Core to Intel Xeon

Figure 6.4: Results for Training and Testing for Cross-Platforms Prediction

decreasing order of prediction accuracy the models are 'etr', 'rf', 'gbr', 'dt', 'xgb', 'knn', 'lr', 'dnn', 'gpr', 'rr', 'svr'. This was observed after taking a majority vote of all three metrics named above. The inferences and analysis from the results for each model are as follows.

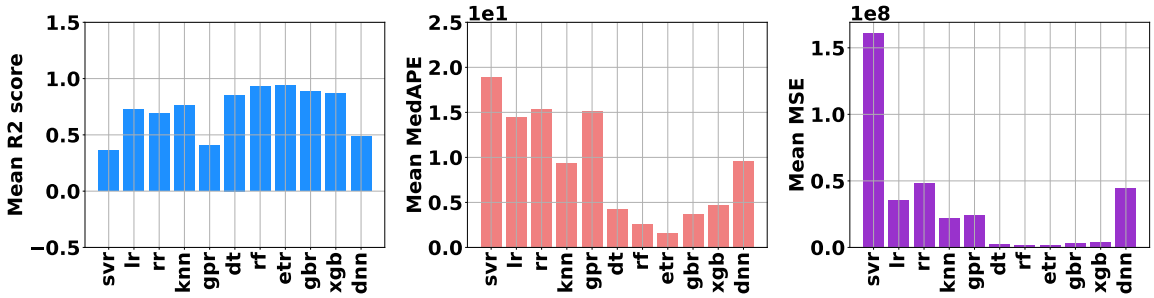


Figure 6.5: Mean Prediction Scores per ML model across all datasets

- ***svr***: Support vector regressor with mean R^2 = 0.36 and mean MedAPE = 18.89% (figure 6.5) 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 R^2 = 0.72 and mean MedAPE = 14.49% and ridge regressor with mean R^2 = 0.69 and mean MedAPE = 15.33% (figure 6.5), both models have nearly similar performance. Low

prediction accuracy in linear models is due to the non-linear relationship between hardware features and runtime (performance).

- ***knn***: KNN regressor with mean R^2 = 0.76 and mean MedAPE = 9.35% (figure 6.5) has better prediction accuracy than kernel svr and linear models. Using a grid search, we found 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 accuracy is that the KNN algorithm is robust to outliers, whereas svr and lr are not.

- ***gpr***: Gaussian process regressor with mean R^2 = 0.40 and mean MedAPE = 15.11% (figure 6.5) still performs better than ridge regressor and kernel svr. The model underperforms in comparison to other models because we believe that the performance datasets may not exactly follow the multivariate gaussian distribution. So, while learning posterior distribution by gpr, it may not be tractable by the model.

- ***dt***: Decision tree regressor with mean R^2 = 0.85 and mean MedAPE = 4.26% (figure 6.5) has 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*** : The random forest (rf) and extra tree regressor (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. The etr model with mean R^2 = 0.94 and mean MedAPE = 1.58% (figure 6.5) performs better than rf model with mean R^2 = 0.93 and mean MedAPE = 2.56%. The reason is due to random splits in etr, which works better in case features are noisy.

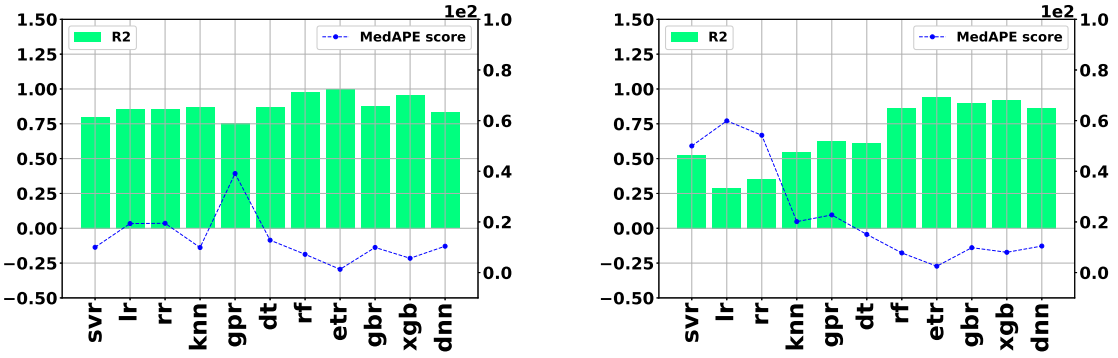
- ***Boosting Models***: Gradient boosting regressor (gbr) with mean R^2 = 0.88 and mean MedAPE = 3.73% (figure 6.5) performs better than extreme gradi-

ent boosting (xgb) with mean R^2 = 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 model is higher than the bagging approaches.

- ***dnn***: Deep neural network (dnn) model (as shown in figure 6.1) with mean R^2 = 0.48 and mean MedAPE = 9.64% (figure 6.5) is not able to outperform other machine learning models except svr and gpr. We believe that dnn model is underperforming due to the small dataset size available for training. The dnn models generally require a large number of data points for learning the mapping function.

6.1.5.5 Estimating the Effect of Performance with respect to Compute-Bound, Memory-Bound and Compute-plus-Memory-Bound Applications

Figures 6.6, 6.7 and 6.8 shows R^2 score and MedAPE on left and right y-axis with ranges $[-0.5, 1.5]$ and $[-10\%, 100\%]$ respectively for compute-bound, memory-bound and compute-plus-memory-bound application types from dataset (refer table 6.1).

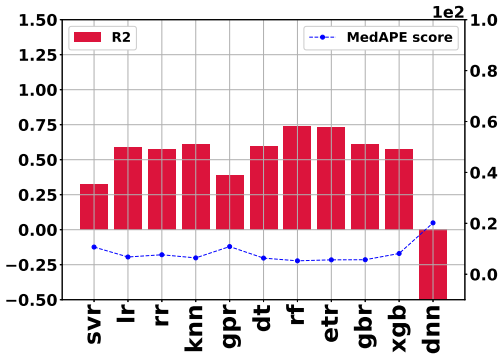


(a) miniFE-physical

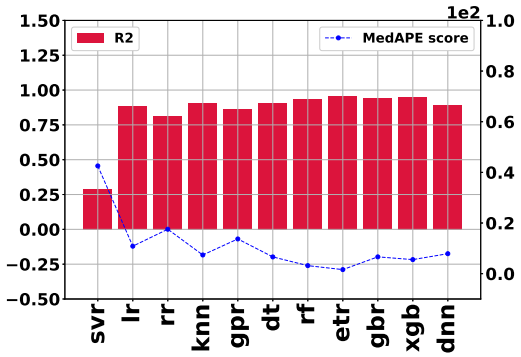
(b) EP-physical

Figure 6.6: R^2 score and MedAPE for Compute-Bound Applications

Each application type plot is color-coded with compute-bound in green, memory-bound in red and compute-plus-memory-bound in violet. The results were compiled for each dataset but we have shown the accuracy of only two datasets per application type. Furthermore, figure 6.9 illustrates the mean MedAPE score for

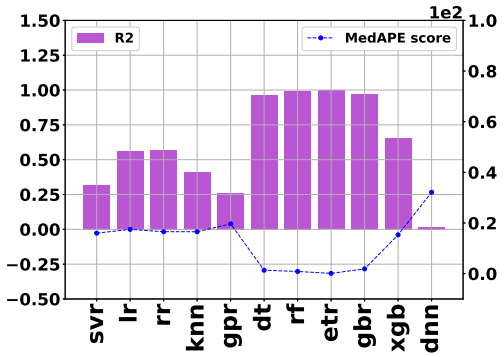


(a) dijkstra-physical

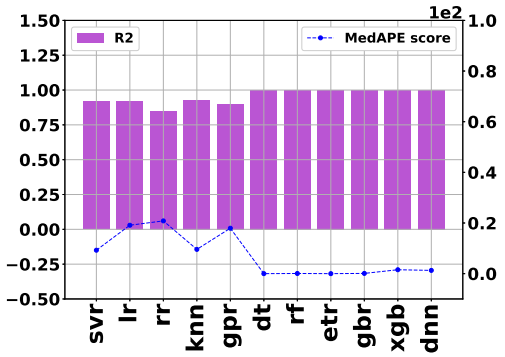


(b) MG-physical

Figure 6.7: R^2 score and MedAPE for Memory-Bound Applications



(a) quicksort-physical



(b) quicksort-simulated

Figure 6.8: R^2 score and MedAPE for Compute-Plus-Memory-Bound Applications

all three application types with scores averaged for all datasets (as listed in table 6.1) for each machine learning model. The first observation from all these figures is that five models, 'dt', 'etr', 'rf', 'gbr' and, 'xgb', are performing well in nearly all scientific applications. This is also the case in plots for the individual application type. Since these models have higher prediction accuracy, we have considered only the result of these five tree-based models for further experiments and the comparison of overall prediction accuracy.

The second observation from figure 6.9 is that on an average taken over MedAPE values of the tree-based models, prediction accuracy in compute-plus-memory-bound applications (with 7.92% mean error) is the highest followed by memory-bound applications (with 9.06% mean error) and the lowest accuracy in the compute-

bound applications (with 9.70% mean error). This observation strengthens the conclusion from the previous section that compute-bound applications have higher runtime variations due to the manufacturer variability in the sophisticated designs of the processors resulting in higher errors. The memory-bound applications have a higher dependence on memory features, which hides the effect of manufacturer variability of processors, causing fewer runtime variations and lower error.

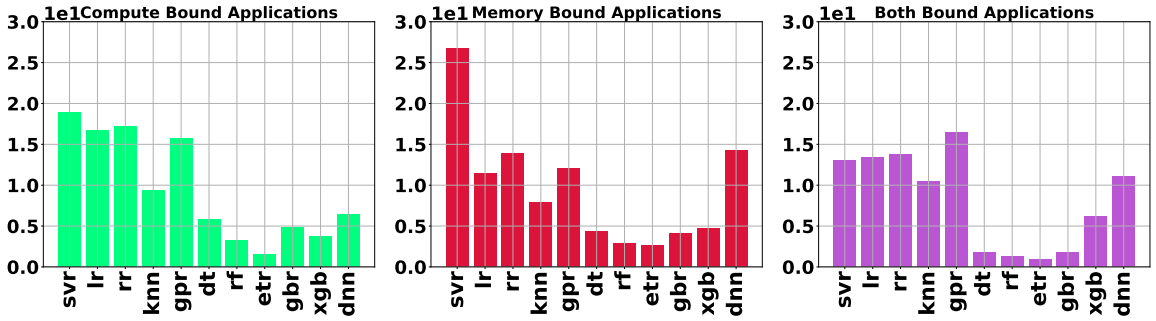


Figure 6.9: Mean MedAPE for all application types

6.1.6 Summary

We demonstrated in this work that using transfer learning, cross prediction in terms of simulated systems to physical systems and prediction between systems with different instruction sets can be achieved with good prediction accuracy. The experiments were performed on thirteen datasets (refer table 6.1) collected by executing five scientific applications on 475 simulated systems built in the gem5 simulator, while eight scientific applications were executed on eight diverse physical computer systems. The conclusions drawn from the above experiments are as follows:

- Dimensionality reduction using PCA is an important pre-processing step and increases the model's prediction accuracy.
- Grid search proved useful for selection of optimal values for hyper-parameters achieving higher prediction accuracy.

- Cross-systems performance prediction from simulated systems to physical systems using transfer learning yields less than 15% error using tree-based models.
- Cross-platform performance prediction from x86 to ARM architectures for simulated systems dataset using transfer learning yields less than 17% error using tree-based models.
- Cross-platform performance prediction in physical systems dataset from Intel Core to Intel-Xeon architectures using transfer learning yields less than 23% error using tree-based models.
- For the performance prediction for simulated to simulated systems 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 have higher prediction accuracy.
- Due to the manufacturer variability in processors, prediction accuracy for memory-bound applications is higher than compute-bound applications.

In the 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 systems to the research community.

6.2 Modeling Performance and Power on Disparate Platforms using Transfer Learning with Machine Learning Models

6.2.1 Overview

Performance has always been an essential factor in the selection of computer systems. However, computing resources' power consumption has emerged as a significant concern in recent times due to environmental hazards. Advancements

in software-hardware co-development have increased the power consumption even further. Therefore, performance and power prediction models using machine learning have become an active area of research. Several researchers have used power prediction to reduce power consumption in computer systems. Work in [85] uses power consumption obtained through prediction as feedback to improve power supply units used in physical computer systems. In [86] and [87] system-level power prediction is performed to understand effect of component-level usage on power consumption. By enabling efficient thread-level parallelism, work in [33] achieves power consumption reduction in heterogeneous cores.

Cross-platform prediction aid us in predicting performance and power from one platform or system to the other. For example, work in [20] shows application performance prediction for GPU from features collected from the x86-based system. Similarly, work in [50] achieves software application performance and power prediction for the ARM-based system by collecting performance counters on an x86-based system using linear regression univariate model. Work in [5] shows that a trained machine learning model from one HPC system can be retrained only on one percent of samples from test (another) HPC system to predict ninety-nine percent of performance data. These research works have focused on cross prediction model with two dissimilar physical systems for training and prediction. In contrast, we explored the transfer learning technique for cross prediction with either training and prediction for the dissimilar instruction set in the case of cross-platform prediction model or prediction for physical systems from simulated systems in the case of cross-systems prediction.

We identified four questions from similar work in the referenced literature. 1) Which of the univariate and multivariate machine learning models predict runtime and power with high accuracy? 2) Which type of application, compute-bound or memory-bound, has higher prediction accuracy? 3) Can we use transfer learning to predict physical system targets (runtime and power) from simulated systems, which we refer to as a cross-systems prediction? 4) Can we predict the runtime and power of one type of physical system (Intel Xeon) from other physical systems (Intel Core) using transfer learning which we refer to as a cross-

platform prediction?

The remainder of the section is organized as follows: Section 6.2.2 describes the computer systems selection and dataset. Section 6.2.3 lists machine learning models and techniques that we have used. Section 6.2.4 provides results of three scenarios: (a) Prediction accuracy analysis of different machine learning models on each application type and system type dataset (b) Evaluation of results concerning compute-bound and memory-bound applications. (c) Prediction accuracy of cross-platform and cross-systems prediction implementation using transfer learning concerning physical and simulated systems. Finally, section 6.2.5 provides concluding remarks and work that we plan to continue.

6.2.2 Dataset Preparation

6.2.2.1 Applications Selection for Workload

We selected applications as workloads for our performance modeling according to their computation and data access patterns categorizing them as compute-bound (CB) and memory-bound (MB). Compute-bound applications such as monte carlo and sha have a higher dependence on processor features than memory features. On the other hand, memory-bound applications like matrix multiplication and dijkstra depend more on memory features than processor features. Some applications, such as quicksort and stitch, depends on both processor and memory features approximately equally categorizing them as compute-plus-memory-bound (CB+MB). Table 6.4 lists the scientific applications and their respective types CB, MB, or CB+MB we have used in our experiments.

6.2.2.2 Computer Systems Selection

First, we built performance datasets by executing the selected applications on simulated systems. For the execution of applications on simulated systems, we built 475 simulated systems in the gem5 simulator [32], a widely accepted simulator for architectural research. Out of 475 simulated systems, 355 systems were built based on x86 instruction set architecture and the remaining 120 systems were based on

Table 6.4: Application and System Types used for Cross Performance and Power Prediction Model with Transfer Learning

Applications	Benchmark	System Type	Data Point	System Type	Data Point	CB/MB
matrix multiplication		Physical	519	Simulated	1780	MB
monte carlo		Physical	260	Simulated	1365	CB
mser	SD-VBS	Physical	52	Simulated	430	CB
stitch	SD-VBS	Physical	52	Simulated	425	MB+CB
svm	SD-VBS	Physical	52	Simulated	390	CB
tracking	SD-VBS	Physical	52	Simulated	425	MB
quicksort	MiBench	Physical	672	Simulated	2730	MB+CB
sha	MiBench	Physical	52	Simulated	367	CB
dijkstra	MiBench	Physical	52	Simulated	362	MB

CB = Compute-Bound (compute-intensive)
MB = Memory-Bound (data-intensive)

the ARM instruction set architecture as discussed in chapter 2.3.1. Table 2.1 shows the hardware feature set (configurations) of all 475 gem5 simulated systems. The hardware feature values for each gem5 simulated system were gathered from real computer systems available today, indicated by the H/W class column. Each application was executed on all simulated systems, as shown in columns three and four in table 6.4. For each execution, we collected runtime from gem5 execution logs and power consumption using McPAT tool [38] using the process explained in section 4.3.2.

Similarly, we built a physical systems performance dataset by executing applications on the physical systems with hardware features set shown in table 6.5. The hardware features from physical systems were collected using the dmidecode utility. Our selection of six systems includes server systems and general-purpose systems commonly used today. To take advantage of hyper-threaded systems, we executed processes from one up to two times the number of cores in the system; that is, we have executed 24 processes on a system with 12 cores. We modified the source code for each application to integrate it with PAPI tool API [95] to collect runtime and power consumption upon execution on each physical system. Both runtime (performance) and power consumption for each application execution on each physical system was collected from application logs.

Table 6.5: Physical Computer Systems for Cross Performance and Power Prediction Model with Transfer Learning

Sr	ISA	CPU Speed GHz	Cores	Mem Type	Mem Access MHz	Mem Size GB	L1-L3 Cache Size*
1	x86	3	2	DDR3	1600	4	32,4
2	x86	3.2	4	DDR3	1600	4	32,6
3	x86	3.2	4	DDR3	1600	4	32,6
4	x86	3.2	4	DDR4	2400	4	32,6
5	x86	3.168	6	DDR4	2666	16	32,12
6	x86	2.4	12	DDR4	1866	16	32,15

*L1 Cache Size is in kB and L3 Cache Size is in MB
Configurations were taken from the following models
1. Intel Core i76500U 2. Intel Core i56500 3. Intel Core i53470 4. Intel Core i56500
5. Intel Core i78700 6. Intel Xeon E52620

6.2.3 Machine Learning Models and Techniques Used

Supervised machine learning algorithms are used to perform empirical performance and power modeling. The machine learning models, use the labeled dataset to learn the function $\Im x$ that maps input X_i to output Y_i. Tables 2.1 and 6.5 depicts processor and memory features for simulated systems and physical systems which defines input X_i for our machine learning models. The output Y_i for our machine learning model has two target variables runtime and power, which we collected during the application's execution on simulated and physical systems. We trained machine learning models to determine the function $\Im x$ by mapping hardware features X_i to runtime and power, the output Y_i. In the prediction phase, the trained model predicts the values of both runtime and power using systems with input features not utilized during training.

6.2.3.1 Models Description

There were two options to build machine learning models considering two output variables runtime and power. The first option was to build a separate machine-learning model for each output called univariate models. The second option was to build a multivariate model where one model predicts both runtime and power simultaneously. We selected different classes of algorithms for machine learning

models such as linear models, boosting models, tree-based models, and neural network models; some are univariate, and others are multivariate models. The scikit-learn library provides linear regression (lr), ridge regression (rr), k-nearest neighbor (knn), and gaussian process regressor (gpr), which can only perform univariate modeling. Linear models lr and rr cannot have a single equation predicting two targets, therefore, implemented as multiple univariate models to predict multiple targets. Furthermore, knn regression returns the weighted average of k-nearest neighbors corresponding to each output separately. On the other hand, the decision tree (dt), random forest (rf), extra tree regressor (etr), and deep neural network (dnn) are multivariate models. Table 6.6 lists all the models that we have used for the experimentation. Figure 6.10 shows the configuration of our deep neural network model with the number of layers, number of neurons used at each layer, and parameter details.

Table 6.6: Machine Learning Models for Cross Performance and Power Prediction Model with Transfer Learning

Models Used	Abbreviations	True Multivariate
Linear Regressor	lr	No
Ridge Regr.[72]	rr	No
K-Nearest Neighbor Regressor[73]	knn	No
Gaussian Process Regressor.[74]	gpr	No
Decision Tree Regressor[97]	dt	Yes
Random Forest Regressor[75]	rf	Yes
Extra Trees Regressor[76]	etr	Yes
Deep Neural Network[98]	dnn	Yes

6.2.3.2 Transfer Learning

Transfer Learning allows the transfer of learning from the already trained model from one problem to another similar problem. The work in [100] describes an excellent example of transfer learning, where knowledge from the pre-trained state of the art BERT model for NLP task is applied to a variety of tasks by just fine-tuning with an additional output layer. We utilized transfer learning for both the cross prediction models, cross-platform and cross-systems prediction. We used transfer learning in cross-platform prediction to train the model on a source

dataset with the systems of one type of instruction-set to predict the targets (run-time and power) with systems of different instruction-set. While transfer-learning is used in cross-systems prediction to predict the targets of physical systems from simulated systems.

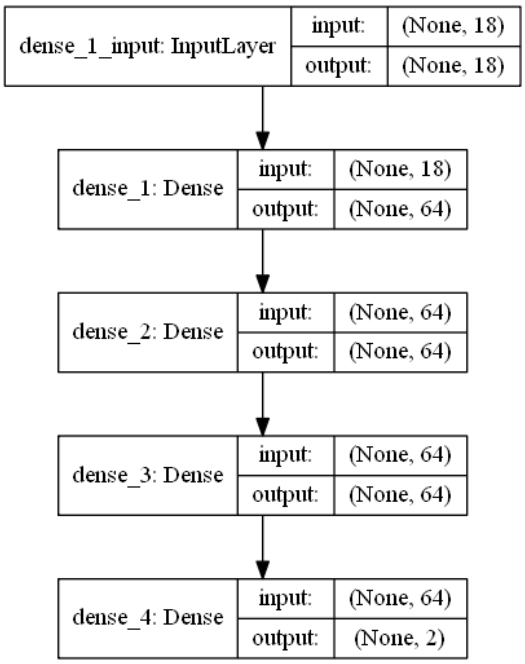


Figure 6.10: DNN Model Summary

6.2.4 Experimental Evaluation

In this section, we have provided the details of experiments performed along with the assessment of the results of univariate and multivariate machine learning models for compute-bound and memory-bound applications on simulated and physical systems' datasets. Furthermore, we also included the experiments for cross-platform and cross-systems multivariate prediction using transfer learning. For using a dataset in machine learning models, we needed to preprocess the dataset. First, we transformed the categorical features memory-type and instruction set architecture with text data to real value using one-hot encoding. We then standardized our real-valued dataset using a $StandardScalar()$ function from the scikit-learn library, which normalizes each feature with a mean of zero and unit variance used in the training and prediction phases of our models.

To ensure that our machine learning model provides high accuracy for any samples selection in the training and prediction phase, we used 10-fold cross-validation using the $ShuffleSplit()$ function with a train-test ratio of 80:20 except for transfer learning experiments. Additionally, we applied grid search for tuning important hyper-parameters to improve the model accuracy. To evaluate the machine learning model's accuracy, we used a R^2 score and a median absolute percentage error (MedAPE). We considered mean values of all 10-folds for R^2 score and MedAPE for prediction accuracy evaluation. The model with higher accuracy has low MedAPE and a high R^2 score. Also, a negative R^2 score implies that the model is not a good fit.

6.2.4.1 Model Comparison for Multivariate Prediction

We trained each machine learning model from table 6.6 with an 80% dataset and tested on the remaining 20% while measuring metric scores for each prediction. We took the mean of runtime and power R^2 score, while MedAPE is used separately each for runtime and power. Mean MedAPE and R^2 score was then combined across all datasets as listed in table 6.4. Figure 6.11 shows the 10-fold cross-validation mean MedAPE and R^2 score for each machine learning model.

6.2.4.1.1 Prediction Accuracy of Univariate Models:

We built two separate machine learning models, one for runtime and the other for power, using four univariate algorithms lr, rr, knn, and gpr using the scikit-learn library. Figure 6.11 shows the prediction accuracy of these models. We observed that the knn model with mean runtime MedAPE = 11.76%, mean power MedAPE = 10.53% and mean R^2 = 0.83 has the lowest prediction errors among the four univariate models due to its ability to map non-linear function between input hardware features and targets, runtime and power. On the other hand, due to the same reason, linear models with algorithms lr and rr have higher errors. The gpr model also has low prediction accuracy because the feature set does not follow gaussian curve resulting in poor prediction of the targets.

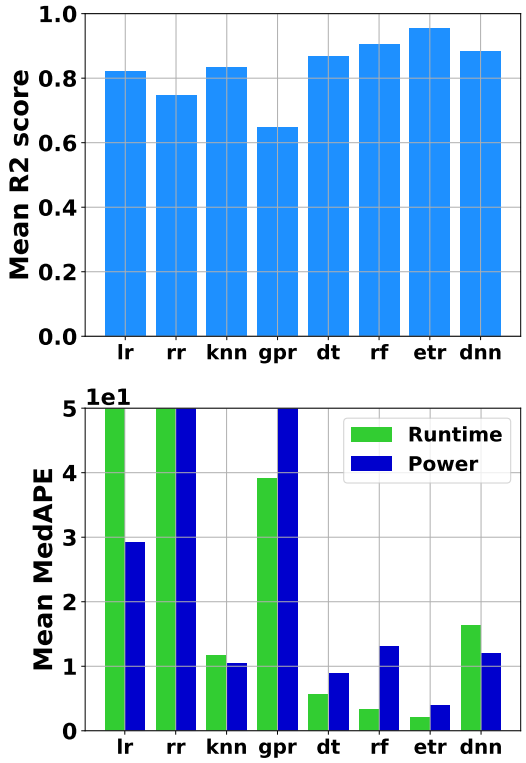


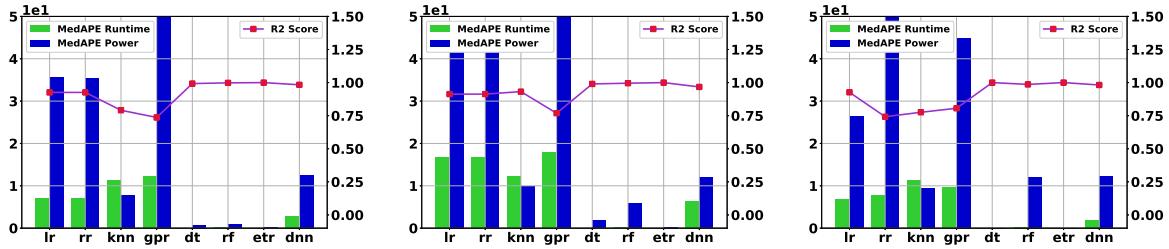
Figure 6.11: Mean of R^2 and MedAPE values per model across all datasets

6.2.4.1.2 Prediction Accuracy of Multivariate Models:

The last four machine learning models dt, rf, etr, and dnn from figure 6.11 are truly multivariate models that predict both runtime and power simultaneously. The dt, rf, and etr are tree-based models, while dnn is a neural network model. From the results in figure 6.11, it is observed that the etr model with mean runtime MedAPE = 2.14%, mean power MedAPE = 3.94% and mean R^2 = 0.95 has the highest prediction accuracy among all other models including dnn. The etr's higher accuracy is due to tree-based models' ability to build decision rules with optimal splitting criterion at each node rather than considering correlation like other linear models. The etr performs better than other tree-based model algorithms 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 due to the availability of a limited number of samples, dnn has a lower accuracy than tree-based models. One important observation is that the multivariate models outperform univariate models.

6.2.4.2 Multivariate Prediction Accuracy per Application Types and System Types

In this section, we first compared the prediction accuracy of the multivariate prediction model for different application types, compute-bound, memory-bound, and compute-plus-memory-bound. Figure 6.12 depicts the example with compute-bound application sha , memory-bound application dijkstra and compute-plus-memory-bound application stitch. Secondly, we compared the prediction accuracy of simulated systems and physical systems.



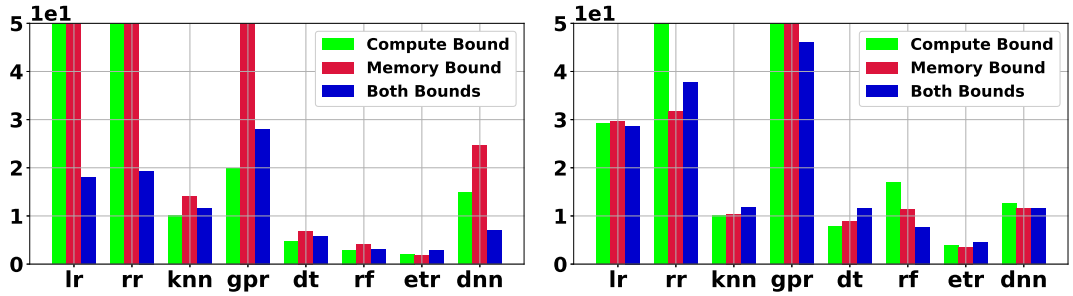
(a) sha (CB) (b) dijkstra (MB) (c) stitch (CB+MB)

Figure 6.12: R^2 score and MedAPE for each of the example Application Types for Simulated Systems

6.2.4.2.1 Prediction Accuracy per Application Type:

We performed multivariate performance and power prediction for applications from table 6.4, and results were averaged over a specific application type, compute-bound (CB), memory-bound (MB) or compute-plus-memory-bound (CB+MB). Figure 6.13 displays the combined mean MedAPE across all applications per application type.

We have three observations from the results. First, the tree-based multivariate models dt, rf, and etr with mean runtime MedAPE = 3.77% and mean power MedAPE = 8.47% outperform all other models, including deep neural network. Second, for tree-based models, the power prediction errors are higher than the runtime prediction errors due to higher variations in power values than runtime values. Third, for non-linear models, the runtime prediction accuracy for compute-bound applications is higher than memory-bound applications due to



(a) Runtime Error

(b) Power Error

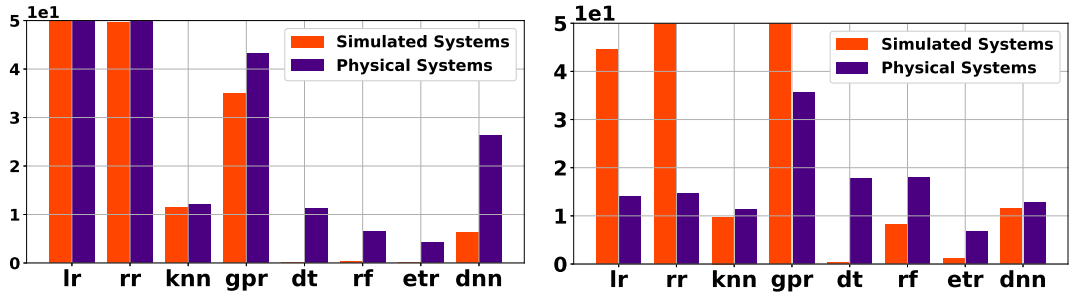
Figure 6.13: Comparing Model Performance with respect to Application Type

the higher number of data points that account for simulation data points. The simulated systems use only one supported out-of-order processor in the gem5 simulator, resulting in lower runtime variations for compute-bound applications. On the other hand, the power prediction accuracy for compute-bound applications is analogous to memory-bound applications because the power consumption of memory units is insignificant compared to processor power consumption, which results in similar power variations in all types of applications.

6.2.4.2.2 Prediction Accuracy per System Type:

Each scientific applications from table 6.4 were executed on both simulated systems and physical systems from tables 2.1 and 6.5 respectively. In this section, we compared the prediction accuracy for both system types, simulated systems, and physical systems. Figure 6.14 shows mean MedAPE for each machine learning model across all applications. We observed that for simulated systems across tree-based models, the prediction accuracy with mean runtime MedAPE = 0.096% and mean power MedAPE = 3.21% is higher than physical systems with mean runtime MedAPE = 7.31% and mean power MedAPE = 14.10%. The lower prediction accuracy in non-deterministic physical systems is because of the resource contention among multiple processes. On the contrary, the simulation environment is deterministic, resulting in fewer variations in simulated systems than physical systems. Even in the case of both system types, simulated systems and physical systems, the power prediction errors are higher than runtime prediction errors due to higher variance in power in line with application types' power and run-

time errors comparison.



(a) Runtime Error (b) Power Error

Figure 6.14: Comparing Model Performance on Physical and Simulated Systems

6.2.4.3 Cross-Platform Multivariate Prediction using Transfer Learning

We carried out cross-platform in both simulation and physical environments. We transfer the learning from the trained machine learning model from the source platform (platform-1) and apply it to the target platform (platform-2). We selected quicksort application to perform cross-platform prediction due to the highest number of simulated systems and physical systems data points with 2730 and 672 respectively among all applications, as shown in table 6.4. In section 6.2.4.3.2, we performed cross-platform prediction in simulation environment by separating data points into three platforms, Intel Core with 1440 data points used as source platform (platform-1) and ARM with 720 data points or AMD with 570 data points one of which used as a target platform (platform-2). Similarly, in section 6.2.4.3.2, we performed cross-platform prediction in the physical environment by separating data points into two platforms, Intel Core with 360 data points used as source platform (platform-1) and Intel Xeon with 312 data points used as a target platform (platform-2).

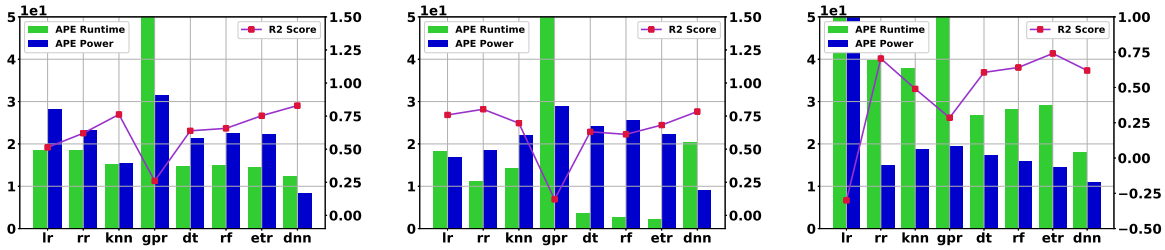
6.2.4.3.1 Cross-Platform Prediction: Intel Core to ARM and AMD

In this section, we use transfer learning to predict targets runtime and power for ARM or AMD systems (platform-2) using a trained model from Intel Core systems' (platform-1) hardware features and both targets in a simulation environment. For transfer learning implementation, we consumed 100% data of platform-

1 and 1% data of platform-2 to train the machine learning model and predicted the targets for the remaining 99% data of platform-2. We trained all models except dnn using combined data of 101% from platform-1 and platform-2 due to the unavailability of partial_fit function for all machine learning models in the scikit-learn library. In the case of dnn, we first trained the dnn model using 100% of the platform-1 dataset, and then we froze all other layers except the last two, which we retrained using 1% of the platform-2's dataset. The results for cross-platform prediction in simulation environment are shown in figures 6.15a and 6.15b. From figure 6.15a, it is observed that for Intel Core to ARM prediction, dnn with R^2 score = 0.83 including runtime APE = 12.47% and power APE = 8.32% outperforms all other models. We also achieved mean runtime APE = 14.24% and mean power APE = 18.61% for dnn and tree-based models combined. From figure 6.15b, it is observed that for Intel Core to AMD tree-based models with mean runtime APE = 2.89% outperform other models, whereas, dnn with mean power APE = 8.98% outperforms other models in case of power prediction.

6.2.4.3.2 Cross-Platform Prediction: 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. For transfer learning implementation, we consumed 100% data of platform-1 and 10% data of platform-2 to train the machine learning model and predicted targets for the remaining 90% data of platform-2. We utilized 10% data from platform-2 for training because the use of smaller than 10% data points resulted in higher errors. We trained all models except dnn using combined data of 110% from platform-1 and platform-2. In the case of dnn, we first trained the dnn model using 100% of the platform-1 dataset, and then we froze all other layers except the last two, which we retrained using 10% of the platform-2 dataset. Figure 6.15c shows the results of cross-platform prediction in the physical environment. We observe that dnn with runtime APE = 18.04% and power APE = 11.08% outperforms all other models.



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

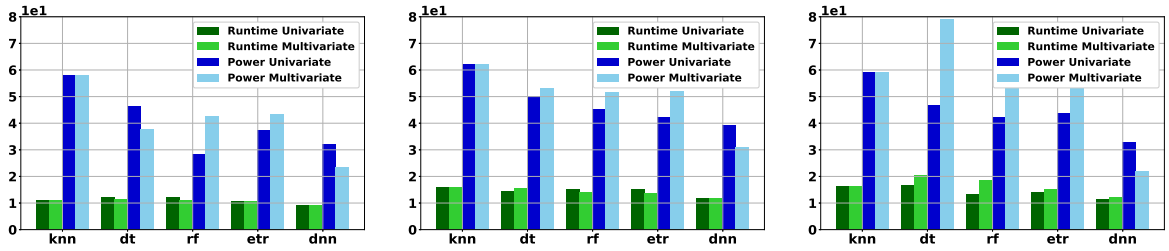
Figure 6.15: Transfer Learning on Quicksort Simulated Dataset (a and b) and Quicksort Physical Dataset (c)

6.2.4.4 Cross-Systems Multivariate and Univariate Prediction using Transfer Learning

In this section, we demonstrate the cross-systems prediction results in which we predict the targets runtime and power of physical systems from the machine learning model trained on the simulated systems' dataset. To compare the prediction accuracy of different application types, we performed cross-systems experiments on svm, a compute-bound application, tracking, a memory-bound application, and stitch compute-plus-memory-bound (CB+MB) application. We performed another experiment for comparing the prediction accuracy of univariate and multivariate models implemented using the same machine learning algorithms for cross-systems prediction. We implemented univariate models using separate machine learning models for runtime and power, whereas we implemented multivariate models using a single machine learning model that predicts both targets. For transfer learning implementation, we utilized a 100% simulated systems dataset and 10% of physical systems dataset for each application to train the machine learning model and predicted targets for the remaining 90% physical system's data. We trained all models except dnn using combined data of 110% from simulated systems and physical systems datasets. In the case of dnn, we first trained the dnn model using 100% of the simulated systems dataset, and then we froze all other layers except the last two, which we retrained using 10% of the physical systems dataset.

6.2.4.4.1 Prediction Accuracy per Application Type:

Figure 6.16 displays runtime and power prediction errors from cross-systems experiment for univariate and multivariate models of the three selected applications svm, tracking and stitch. It is observed from figures 6.16a, 6.16b and 6.16c that for all the three application types compute-bound (svm), memory-bound (tracking) and compute-plus-memory-bound (stitch) dnn multivariate model outperforms all other models with svm runtime APE = 9.10% and power APE = 23.38%, tracking runtime APE = 11.81% and power APE = 31.03% and stitch runtime APE = 12.11% and power APE = 22.05%. We also observed that runtime errors between univariate and multivariate dnn implementations have a similar range, whereas power errors are lower in multivariate dnn implementation.



(a) svm (CB) (b) tracking (MB) (c) stitch (CB+MB)

Figure 6.16: APE error for Simulated Systems to Physical Systems Prediction using Transfer Learning

6.2.4.4.2 Prediction Accuracy of Univariate and Multivariate Implementation:

Figure 6.16 shows the results from the univariate and multivariate implementation of selected machine learning models. We observed from the plots that the univariate and multivariate implementations of dt, rf, etr, and dnn have different prediction accuracy categorizing them as multivariate models. On the other hand, knn univariate and multivariate implementation gives precisely the same prediction accuracy strengthening the argument that knn is not truly a multivariate model.

6.2.5 Summary

In this section, we experimented with machine learning models built from various classes of algorithms for a wide variety of datasets consisting of three different application types and two different system types. We compared the results and made important observations to give an in-depth analysis of multivariate performance and power modeling. Furthermore, we also utilized the transfer learning approach for cross-platform and cross-systems predictions. The conclusions from the experiments performed are as follows:

1. Multivariate tree-based models namely dt, rf and etr outperformed all other models including dnn models. Also, multivariate models performed better than univariate models if the training and prediction are made on the same application dataset.

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

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

4. Multivariate prediction errors in simulated systems were less than physical systems due to the deterministic nature of simulated systems.

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

6. For cross-systems prediction, dnn multivariate outperformed all other univariate 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 datasets to the research community. We also plan to experiment with the data augmentation technique to add more data to the physical systems dataset.

CHAPTER 7

Cross Prediction Models: Scaling vs. Transfer Learning

7.1 Overview

Computer systems come with many options today, each having diverse hardware features due to technological advancements in a processor, memory, and other components. These hardware feature differences result in different software execution performance (runtime) on various computer systems; therefore, selecting a computer system with optimum performance for a given software is desirable. However, access to several physical systems having dissimilar hardware features is a difficult task. To overcome this problem, we have already proposed a solution that we called "cross performance prediction with scaling" in our previous work [47]. In which, we predicted the performance of physical systems using a machine learning model trained only on the performance dataset of simulated systems built using the gem5 simulator [32].

Our selection of the gem5 simulator for building simulated systems is because it is a cycle-accurate simulator [89] with support of full-system mode and system-call emulation mode. Work in [31] has shown that the gem5 system built with full-system simulation mode provides accurate software performance compared to the physical system. However, full-system setup for each computer system is arduous and slow as compared to native execution according to [39] and [40]. On the other hand, the system-call emulation mode of gem5 provides systems' simulation at a higher speed with lower accuracy. Therefore, our goal is to per-

form accurate performance prediction of physical systems using gem5 simulated systems built using system-call emulation mode.

Fast and accurate performance prediction for hardware/software co-development is very much desirable. The task is even more challenging when access to required physical computer systems or platform is unavailable. Cross-platform performance prediction has been widely used to solve this challenging problem. The cross-platform performance prediction approach uses a machine learning-based model trained on the features from the source platform to predict the target platform's performance which is different from the source platform. This method allows us to predict the software's performance without actually running it on the target hardware platform.

A LASSO machine learning model was recently proposed to predict the ARM-based target system's performance by collecting performance counters on x86-based systems [3] [50] for several MiBench and SD-VBS benchmarks. Similarly, the P4 framework has been proposed [4] for ARM-based system's prediction using a neural network trained using performance counters that identify application phases on the x86-based system for heterogeneous systems. The cross-platform framework was proposed [20] for the prediction of GPU performance from x86-based systems. A method was proposed [21] to select the cloud's optimum configuration for a user application using cross prediction that predicts the application's performance on a cloud with different configurations using platform-independent features collected from general-purpose systems.

The goal for these research works is to use cross-platform prediction to predict a target physical's performance from the source physical system using machine-learning performance modeling. On the other hand, our goal is to use cross performance prediction to predict physical systems' performance from simulated systems using the decision-tree machine learning model, which is a different goal [47].

It is expected to use the dataset with the same feature space in traditional machine learning models, possibly with the same distribution. The same feature space samples are then divided into train and test sets to be used by the machine

learning model for training and prediction phases for performance modeling. On the other hand, in transfer learning, a machine learning model is trained from feature space from one task and performs prediction for different but similar tasks with different feature spaces.

Several researchers have used transfer learning for cross-platform prediction. For example, work in [23] applied transfer learning to predict the runtime of a target server C with features and runtime from source servers A and B. They modify the benchmark applications code to count the number of loops, assignment, conditional, and message passing instructions. They use these counters as input features and output runtime from servers A and B to train the machine learning model. During the training, they also use 1% to 10% features from target server C and then prediction for the remaining 90% features prediction is performed.

Similarly, work in [5] used transfer learning for cross-platform performance prediction between two HPC-systems p1 and p2. They collect features such as the number of nodes, number of processes, and application-specific features of Mantevo miniFE and miniMD mini-applications along with runtime. They then train the machine learning model using the feature space of source HPC system p1 and only 1% of target HPC system p2 to predict the remaining runtimes for target HPC system p2.

In the transfer learning implementation of the research works in [5] and [23], a small percentage of the dataset from the target system is used in training machine learning model for cross performance prediction. Our goal is to train the machine learning model with the source system's feature space having similar characteristics but not the same feature space as the target system. In this work, we further extended our previous work to performance cross performance prediction using transfer learning by leveraging the said ability of transfer learning. To use transfer learning for cross performance prediction models, we trained a machine learning model on simulated system's performance, and source physical system's performance datasets combined and predicted the different target physical system's performance. Our consideration is that source system access is available with dissimilar hardware features from the target physical system.

We outline our contributions from this work below:

- We used the previously proposed model "Cross Performance Prediction with Scaling" that utilizes scaling factors to predict physical systems' performance from simulated systems.
- We proposed a new model, "Cross Performance Prediction with Transfer Learning," in which a combined performance dataset from simulated systems and a source physical system predicts the performance of the target physical system, which is different from the source system.
- We assessed the prediction accuracy individually for both the variants of cross performance prediction models using applications from different application domains having different computation and data access patterns from San Diego Vision Benchmark (SD-VBS) and MiBench suites.
- In the cross performance prediction model with transfer learning, we compared the prediction accuracy from each model trained with different source physical systems for a given target system to assess the source system having the highest prediction accuracy for most of the applications.
- We also developed a methodology to extract the set of rules (boolean logic) from binary tree build during the decision tree's training to analyze how the runtime value (performance) is predicted for a given set of hardware features.

The remainder of the chapter is organized as follows: Section 7.2 articulates the related work using cross prediction and transfer learning. The section describes the cross performance prediction model. Section 7.3 describes both variants of our cross performance prediction model, the one with scaling and the other with transfer learning. Section 7.4 describes the procedure to build simulated systems, applications selection for workload, and physical systems selection for cross performance prediction. It also details the decision tree regression, a machine learning model that we have used for our cross performance prediction models. Section 7.5 and 7.6 articulates the performance dataset construction in experimen-

tal details and detailed analysis from results from our cross performance prediction models. Section 7.7 compares the prediction accuracy between both models, cross performance prediction with scaling, and cross performance prediction with transfer learning. Finally, section 7.8 has the concluding remarks and tasks that we plan to continue in future work.

7.2 Related Work

Performance modeling research dates back many years. In particular, cross performance prediction has proliferated due to the availability of diverse hardware platforms. In this section, we discuss several research works that performed cross performance prediction.

7.2.1 Cross-Platform Performance Prediction

Several research works [3] [4] [20] [50] [51] have focused on cross-platform performance prediction to predict the performance of the target architecture from a performance dataset collected from source architecture. In some cases, source and target architecture themselves are different, whereas in some cases, architectures are the same, but systems have different processors and memory features.

Cross-platform prediction is used in [50] to show that the performance of a target ARM-based system can be predicted from a source x86-based system. They collect performance counters from x86-based host systems to be used as input features in the training phase. To use the actual runtime (performance) in the training phase, they collect the runtime from target ODROID-XU3 ARM-based systems for 157 different application programs from the ACM-ICPC database. A LASSO machine learning model is then trained using performance counters from x86-based source systems and the actual runtime from the ARM-based target system. In the prediction phase, they collect only the performance counters of 35 applications from Mi-Bench, SD-VBS, and SPEC 2006 benchmarks from x86-based host systems. The performance counters are provided as input features to the trained machine-learning model to predict these benchmarks applications' performance

on an ARM-based target system without actually executing them on the target system. The work achieves an average prediction error of about 1.4% with mser 1.8%, svm 2%, tracking 1%, stitch 1.9%, dijkstra 0.75% and sha 1%. Our cross performance prediction model differs in two ways. First, our model predicts the runtime of physical systems from the simulated systems, while referenced work uses only physical systems for both training and prediction. Second, work in [50] uses performance counters as input features for the machine learning model; on the other hand, we use hardware features values (cpu clock, ISA, cache size, memory type, access speed, size, etc). The use of performance counters captures the application's dependence on the hardware features, but it requires the execution of the application on the source systems to collect performance counters, while our method does not require execution to collect hardware features used for prediction. These two reasons cause the prediction errors of our implementation to differ from the referenced work.

Similarly, the work in [4] applies cross-platform prediction for predicting the performance of a chosen target platform from a different source platform. They have used four different platforms mobile Odroid XU3 ARM-based platform and the remaining three x86-based, server Intel SR1560SF, server Sun Fire X4270 and server Dell PowerEdge R810. They have collected 11 and 12 performance counters from ARM-based platforms and x86-based platforms for 129 industry-standard benchmark applications. They capture the performance counters that collect the event for different phases of the benchmark application. Then the neural network model is trained using these performance counters as input features to predict the performance. Work in [4] performs cross prediction between four different systems; however, they are all physical systems. Also, the author uses performance counters for training and prediction of the machine learning model. On the other hand, our cross performance prediction model predicts physical systems' performance from simulated systems using directly processor, cache and memory hardware features.

Cross-platform prediction is even used to predict GPU performance from CPU-based software applications in [20]. First, they execute the CPU version applica-

tion on a host with Intel Xeon Processor E3-1241 v3 and extract or calculate the features from each runs that affect the GPU performance. They extract features like ILP, the number of memory or control instructions, cold references, reuse distance. They also calculate some features like cache bank conflict, global memory coalescing, branching pattern, floating-point multiplication, division, exponential operations. They train the ensembled machine learning model using these features. The prediction is performed for two GPU platforms, Maxwell GTX 750 and Kepler GTX 660 Ti 10 benchmark applications.

7.2.2 Performance Prediction with Transfer Learning

In general, the machine learning model assumes that training and testing samples are sourced from the same dataset; therefore, they have the same feature space and distribution. A new dataset will result in training a machine learning model from scratch to learn the new relationship. However, transfer learning allows retraining of an already trained machine learning model with a small percentage of new datasets resulting in improving the learned knowledge with the new dataset. The survey paper [22] has provided in-depth knowledge of different transfer learning categories and their implementation. Transfer learning is widely used in performance prediction to predict the different sources and target systems' performance with similar features.

Transfer learning is used in [23] for automated performance prediction of parallel applications with a message passing interface (MPI). The work collects domain-independent runtime features from three MPI-based benchmarks Graph500, GalaxSee, and SMG2000, by instrumenting the code for counting loops, conditions, assignment, and communication messages. Runtime and features are collected to build performance data by executing instrumented benchmarks on three servers A, B, and C having different hardware features, including network. The random forest machine learning model is trained on performance data using source servers B and C plus 1 to 10% of server A's performance data. The trained model predicts the remaining server A's performance data with an accuracy of about 10% to 20% for Graph500 and SMG2000 benchmarks but the higher error in the case of

GalaxSee. Work in [23] uses transfer learning to train the model using 100% performance data from two source physical systems to predict for one target physical system. On the other hand, we use transfer learning to train the model from simulated systems for the performance prediction of physical systems.

ModelMap, a novel instance-based transfer learning technique, is proposed in [25]. In this work, performance in terms of response time of the MariaDB database is predicted using various request types of the TPC-C benchmark. Several virtual machines were built using CPU quota, disk I/O quota, and database buffer pool quota to create systems with dissimilar system configurations and database size changes. Database response time was measured on each of the virtual machines with different configurations and database sizes. To implement transfer learning, they started the machine learning model with empty sample space, and with each system or database change, new samples in the range of 0 to 5% were fed incrementally to an already trained machine learning model. The trained machine learning model with incremental data predicts the performance of database software with a new set of samples with high accuracy when trained with sufficient data.

Performance predicted from dissimilar systems' configurations is always of interest because we can select the system with the best configuration providing optimum performance. Generally, a black-box model collects and predicts data from a real system for performance prediction, but performance measurement may not scale according to system configuration changes. Therefore, authors of [26] proposed a cost-aware transfer learning model that predicts the performance of the real system from data from other sources such as simulation. The experimentation is performed on a robotic system, a NoSQL database, and stream processing applications.

Similarly, the transfer learning model in [5] shows that a trained machine-learning model from one x86-based system's performance data can be retrained only on one percent performance data from different x86-based systems to predict the remaining ninety-nine percent data. Experiments are performed on two IBM Blue Gene and two Cray systems using Mantevo mini-applications miniMD,

miniAMR, miniFE, LAMMPS as workloads. The work achieved an accuracy of about 10% for the miniFE application. Similar to [23], work in [5] also uses transfer learning for physical HPC systems, whereas we use transfer learning between simulated systems and physical systems.

7.3 Cross Performance Prediction Models

In this work, we utilized two cross performance prediction models, the cross performance prediction model with scaling and cross performance prediction model with transfer learning. In our earlier work [47], we already introduced cross performance prediction model with scaling as shown in figure 7.1. We introduced a new cross performance prediction model that uses the transfer learning technique to achieve the same goal, as shown in figure 7.2. This section describes the three phases of both models; training, cross performance prediction, and model evaluation.

7.3.1 Cross Performance Prediction with Scaling

Our cross performance prediction model with scaling used only a simulated systems' performance dataset to train the machine learning model. The trained machine learning model was then provided with the physical system's hardware features to predict the runtime. However, the physical system's predicted runtime vastly differs from the actual runtime because the machine learning model trained only on a simulated systems performance dataset was used to predict the runtime. Therefore, we applied the scaling factor, as explained in sections 5.1.2 and 5.2.2.

In the training phase, we selected hardware features from M simulated systems $Xs_j where j, 1 \leq j \leq M$ and also collected the actual runtimes $ys_j \in \mathbb{R}$ by executing each selected applications. We then encoded text features Xs_j into real-valued features set $Xs'_j \in \mathbb{R}^d$. The machine learning was then trained using Xs' and ys to learn a function such that $\Im(Xs'_j) \approx ys_j \forall j$ called "Learned Model". In the cross performance prediction phase; first, we collected hardware features Xp_i of

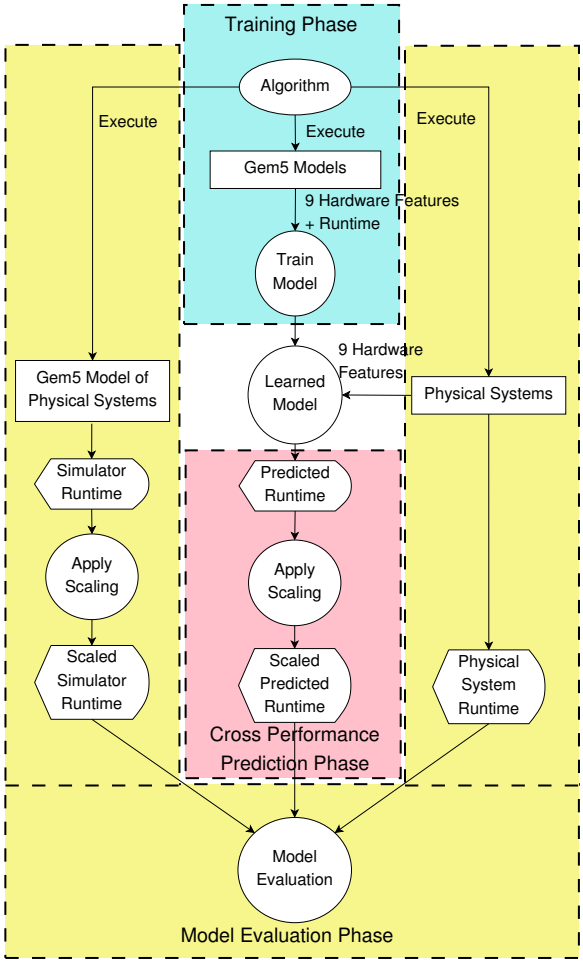


Figure 7.1: Cross Performance Prediction Model with Scaling

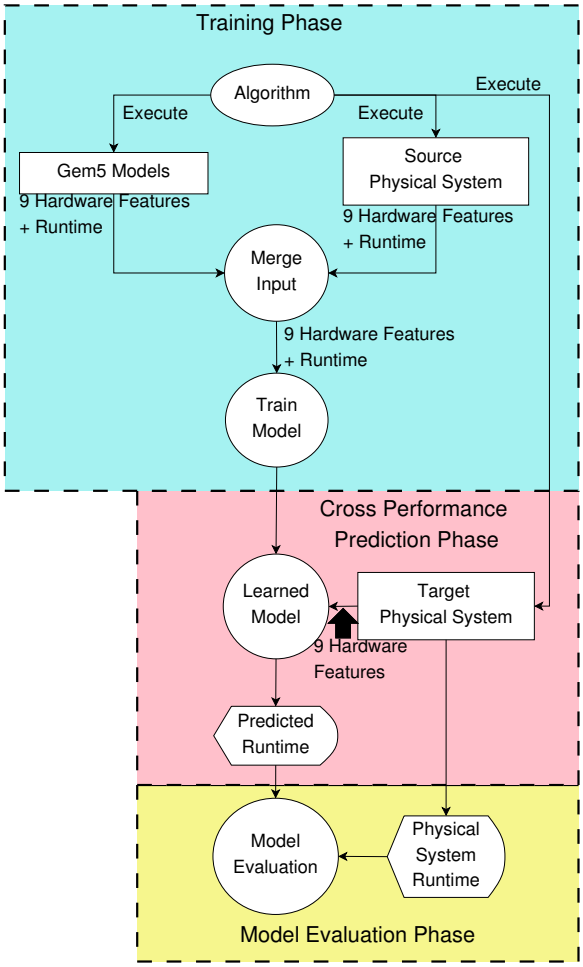


Figure 7.2: Cross Performance Prediction Model with Transfer Learning

each of the N physical systems $i, 1 \leq i \leq N$. After encoding textual features from Xp_i into $Xp'_i \in \mathbb{R}^d$, we provided Xp'_i to "Learned Model" to predict the runtime $yspred_i$ of the physical systems using learned relationship from the training phase $\Im(Xp'_i) \rightarrow yspred_i$.

In the model evaluation phase, we applied the scaling factor to the cross predicted runtime $yspred_i$ of physical systems which is a combination of a major factor and a minor factor. The major factor measures the gap between hardware design between physical systems and the gem5 simulated system built with a system-call emulation mode. The minor factors are the variations in the major factor caused by compute-bound, memory-bound and compute-plus-memory-bound applications' dependency on processor, cache and memory features of each system. We determined major factor, minor factor and scaling factor as explained

in chapter 5. We used the scaling term in this approach because it indicates that predicted runtimes are scaled (adjusted) using scaling factors to estimate physical systems' actual runtime accurately.

7.3.2 Cross Performance Prediction with Transfer Learning

We show in the scaling model that when a machine learning model is trained from only the simulation systems' performance dataset, the predicted runtime is vastly different from the actual physical system's runtime. Our cross performance prediction model with transfer learning used transfer learning instead of scaling technique to improve the accuracy of prediction by providing additional knowledge to the machine learning model. The machine learning model gains additional knowledge by utilizing a small percentage of the accessible physical system's performance dataset denoted as source physical systems in addition to the simulated systems performance dataset during the training phase. In the training phase, first, we collected hardware features and actual runtimes from simulated systems (M) denoted as $Xs_j werej, 1 \leq j \leq M$, and $ys_j \in \mathbb{R}$ respectively by executing selected applications. Similarly, we also collected hardware features and actual runtimes from the source physical systems, $Xpsrc$, and $ypsrc$. We then merged the hardware features Xs and $Xpsrc$ into $Xsrc$ and actual runtimes ys and $ypsrc$ into $ysrc$. We encoded features with text data from $Xsrc$ into $Xsrc' \in \mathbb{R}^d$ to form a dataset with real numbers only. For example, systems with nine features represented as $Xsrc' \in \mathbb{R}^9$. The machine learning model is then trained using $Xsrc'$ and $ysrc$ to learn a function $\Im(Xsrc') \approx ysrc$ that maps hardware features values to actual runtimes such that error between actual and predicted runtime is minimum for all samples. We referred to the trained machine learning model as the "Learned Model."

The cross performance prediction phase of the cross performance prediction model with transfer learning used a different set of physical systems unseen during the training phase referred to as target physical systems. Ideally, we assume that the target physical systems are inaccessible requiring prediction. We collected hardware features from target physical systems $Xptrg$ and encoded the

textual data from $Xptrg$ to $Xptrg'$. We also collected actual runtimes from target physical systems $yptrg$ required to analyze the prediction accuracy. The features $Xptrg'$ of target physical systems were used as input features to "Learned Model," a trained machine learning model, to predict the runtimes for target physical systems $yptrgpred$ using learned relationship from the training phase $\Im(Xptrg') \rightarrow yptrgpred$. We performed several predictions of performance for the same target system each time with a different source system used during training, which allowed us to select the source system that provides the optimal runtime prediction for a given target system. In the model evaluation phase, we compared actual runtime $yptrg$ with a set of predicted runtimes $yptrgpred$ predicted using a disparate set of source physical systems for each target physical system.

7.4 Experiment Platforms

This section provides detail of building gem5 simulated systems, articulates application selection for workloads, and provides information about physical systems used for cross performance prediction.

7.4.1 Simulated Systems

As detailed in chapter 2.3.1, we built 475 systems in gem5 simulator using system-call emulation mode with features listed in table 2.1 that were collected from real-world systems. We modified the gem5 source to support third-level cache and add several memory types to represent the simulated systems more accurately compared to the real systems. However, the limitation of these systems is the use of a single out-of-order processor model supported in gem5.

7.4.2 Applications Selection for Workload

Our selection of applications as the workload is shown in table 7.1 includes four applications mser, svm, tracking, and stitch from San Diego Vision Benchmark (SD-VBS) [70] and two applications sha and dijkstra from MiBench [63] bench-

Table 7.1: Applications used as workloads used for Comparison of Scaling versus Transfer Learning Cross Performance Models

Application	Benchmark	Intensiveness
mser	SD-VBS	compute-intensive
svm	SD-VBS	compute-intensive
tracking	SD-VBS	data-intensive
stitch	SD-VBS	compute-intensive and data-intensive
dijkstra	MiBench	data-intensive
sha	MiBench	compute-intensive

compute-intensive = compute-bound
data-intensive = memory-bound

marks. These applications represent different application domains; svm is a machine learning algorithm, mser, stitch, tracking are image operations, sha is an encryption algorithm, and dijkstra is a network protocol. We selected these specific applications due to their known computation and data access patterns. According to [70], mser and svm depend more on computation, placing them in the compute-bound category; tracking depends more on data access, categorizing it as memory-bound, whereas stitch depends on both computation and data access (compute-plus-memory-bound). Similarly, sha is compute-bound, where dijkstra is memory-bound. The compute-bound applications depend more on processor features because they are compute-intensive, whereas memory-bound applications depend more on memory features. We have demonstrated in our previous work [47] that we can identify the dominant processor or memory features for each application using Pearson Correlation Coefficient [94] between runtime and hardware features.

7.4.3 Physical Systems for Cross Performance Prediction

We limit the scope of our cross performance prediction work to x86-based systems, mainly using Intel processors. Our selection of four physical systems includes a server system, a high-end system, and general-purpose systems. The server system has 12 cores Intel Xeon E5-2620 v3 processor with 16GB of DDR4 memory, the high-end system has six cores Intel Core i7-8700 processor with 16GB DDR4 memory, and the two general-purpose systems have two cores Intel Core

i5-6500 processor and four cores Intel Core i7-6500U with DDR3 memory. We utilized the same nine features values for physical systems, as shown in table 7.2, which we collected using the dmidecode utility.

Table 7.2: Physical Computer Systems used for Comparison of Scaling versus Transfer Learning Cross Performance Models

Sr	ISA	CPU Speed GHz	Cores	Mem Type	Mem Access MHz	Mem Size GB	L1-L3 Cache Size
A	x86	3.2	4	DDR3	1600	4	32kB,6MB
B	x86	2.4	12	DDR4	1866	16	32kB,15MB
C	x86	3	2	DDR3	1600	8	32kB,4MB
D	x86	3.2	6	DDR4	2666	16	32kB,12MB

Configuration taken from following models:
A.Intel Core i5-6500 B.Intel Xeon E5-2620 v3 C.Intel Core i7-6500U D.Intel Core i7-8700

7.4.4 Decision Tree Regression Machine Learning Model

Decision tree regression is one of the popular machine learning models due to its simplicity and intelligibility [101] [102]. Furthermore, in a separate study performed by us in [42], we demonstrated that decision tree regression has higher prediction accuracy than other machine learning models for our performance dataset. Therefore, we employed scikit-learn [65], a python-based library implementation of decision tree regression [6] [90] as our machine learning model for our cross performance prediction.

Given the input features x_i from input feature set $x = x_0, x_1, ..., x_n$ and target value y, decision tree regression model recursively partitions feature space of x_i such that samples with close target value are on the same side of the tree. For regression problems, mean squared error (MSE) identifies the closeness between actual and target values and is given by:

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \bar{y_i})^2 \tag{7.4.1}$$

Let data of node m is represented by Q. At each of the decision tree node m, there could be several possible partitions in data set Q. For each candidate

partition $\theta = (x_i, t_{x_i})$, data is partitioned into $Q_{left}(\theta)$ (left sub-tree) and $Q_{right}(\theta)$ (right sub-tree) (equation 7.4.2), where x_i is compared against the threshold t_m.

$$Q_{left}(\theta) = (x, y)|x_i <= t_m \tag{7.4.2a}$$

$$Q_{right}(\theta) = Q \setminus Q_{left}(\theta) \tag{7.4.2b}$$

For each candidate partition, mean squared error (MSE) is calculated for left and right sub-trees, which is then used to calculate the impurity function given by

$$G(Q, \theta) = \frac{n_{left}}{N_m} MSE(Q_{left}(\theta)) + \frac{n_{right}}{N_m} MSE(Q_{right}(\theta)) \tag{7.4.3}$$

Out of all the candidate partitions, select the one which minimizes the impurity

$$\theta^* = argmin_{\theta(x_m, t_m)}\, G(Q, \theta(x_i, t_i)) \tag{7.4.4}$$

At the root node, candidate partitions consist of one partition for each input feature $x_i|x_i \in x$ input feature set and threshold t_{x_i}. For each of the candidate partitions, impurity function $G(Q, \theta)$ is calculated using mean square error. Partition with smallest impurity is selected as criteria $\theta = (X_i, t_m)$ for this node. Based on criteria $\theta = (X_i, t_m)$, all the samples Q from root node with $x_i <= t_m$ are partitioned into $Q_{left}(\theta)$ and $x_i > t_m$ are partitioned into $Q_{right}(\theta)$. Recursively partition the subsets $Q_{left}(\theta)$ and $Q_{right}(\theta)$ until maximum allowable depth of the tree is reached $N_m < min_{samples}$ or sample size has reached to one $N_m = 1$, that means, mean squared error cannot be reduced further. Each of the non-leaf nodes is decision points that guide the direction in which the data path is traversing for a set of specific feature values of a sample until a leaf node is reached which provides the target (predicted) value y for that sample. Mean of y values is considered as the target value for the leaf node with more than one samples.

7.5 Cross Performance Prediction with Scaling

In this section, we discuss the experiments carried out for cross performance prediction with scaling. First, we provide experimental details about the construction of performance datasets using simulated systems and physical systems. We then articulate the results from performance prediction using machine learning model trained on the simulated systems performance dataset. We analyze the prediction accuracy results of our machine learning model for different applications. Finally, we outline the procedure for cross prediction and show the prediction accuracy results with or without scaling factor.

7.5.1 Experimental Details

First, we built the performance dataset from simulated systems. To built the simulated systems' performance dataset, we collected the values of nine hardware features $Xs \in \mathbb{R}^9$ and actual runtimes $ys \in \mathbb{R}$ by executing each selected application from table 7.1 on each of the 475 gem5 simulated systems from table 2.1. A machine learning model does not accept textual data; hence, we employed one-hot encoding to convert text data features, instruction-set-architecture (ISA) and memory type, to real-valued features $Xs' \in \mathbb{R}^9$. Therefore, the simulated systems performance dataset consists of all real-valued hardware features and the corresponding actual runtime $[Xs', ys]$ of an application executed on each of the 475 simulated systems.

In our previous work [45], we have shown that a 60:40 train-test split ratio provided much higher prediction accuracy in the simulation-based prediction model. Therefore, we selected 60% samples from simulated systems performance dataset $[Xs'_j, ys_j], 1 \leq j \leq M, M = numberoftrainingsamples$ for the training phase and the remaining 40% $[Xs'_k, ys_k], 1 \leq j \leq N, N = numberoftestingsamples$ was used for prediction (testing). We used 5-fold cross-validation to ensure high prediction accuracy for each fold by random samples selection of training and testing using the ShuffleSplit() function from the scikit-learn library. We trained the decision tree regression, a machine learning model on training samples $[Xs'_M, ys_M]$

for each application's simulated systems performance dataset separately, which we referred to as "Learned Model."

For the cross prediction phase, we collected hardware features from the four physical systems $Xp_i \in \mathbb{R}^9, 1 \leq i \leq 4$ as shown in table 7.2. We built gem5 simulated systems using physical systems' feature values Xp and collected actual runtime ysp by executing applications on simulated physical systems. We also encoded the text features ISA and memory type from Xp using the same one-hot encoder class used by simulated systems performance dataset to apply the same encoding to the physical systems' features resulting in real-valued physical feature set Xp'. We also collected actual runtimes yp from physical systems by executing applications on all four physical systems. To reduce the effect of variations in the non-deterministic physical system's runtime, we collected 40 runtime values by executing each application on each physical system 40 times and the mean of these 40 runtime values were used as the actual runtime. We then provided an encoded physical system's hardware features Xp' as an input to the learned model to predict the physical system's runtime $yspred$ for cross performance prediction. We analyzed the prediction accuracy between the actual runtime ysp from simulated physical systems and the predicted runtime $yspred$ as well as accuracy between the actual physical system's runtime with unscaled and scaled cross predicted runtime $yspred$

7.5.2 Results

In the results section for cross performance prediction with scaling, we evaluate our model in two ways. First, we evaluated the decision tree model's prediction accuracy by studying the rules that determine predicted runtime for a given application on a specific system. Second, we applied the scaling factor determined from our previous work and analyzed the actual physical system's runtime results with unscaled and scaled cross-predicted runtime from the model ("Learned Model"), trained from simulated systems dataset.

7.5.2.1 Prediction Accuracy of Decision Tree Machine Learning Model

We plotted actual runtime (*ysp*) from the simulated physical system and predicted runtimes (*yspred*) for each of the four physical systems in figure 7.3. Considering the percentage errors of all four physical systems shown in table 7.3, mean percentage error for svm, mser, tracking, stitch, sha, and dijkstra applications are 2.9%, 5.17%, 3.87%, 1.77%, 10.93%, and 9.98% respectively. Hence, we achieved a prediction accuracy of at least 90% for each application between *ysp* and *yspred*.

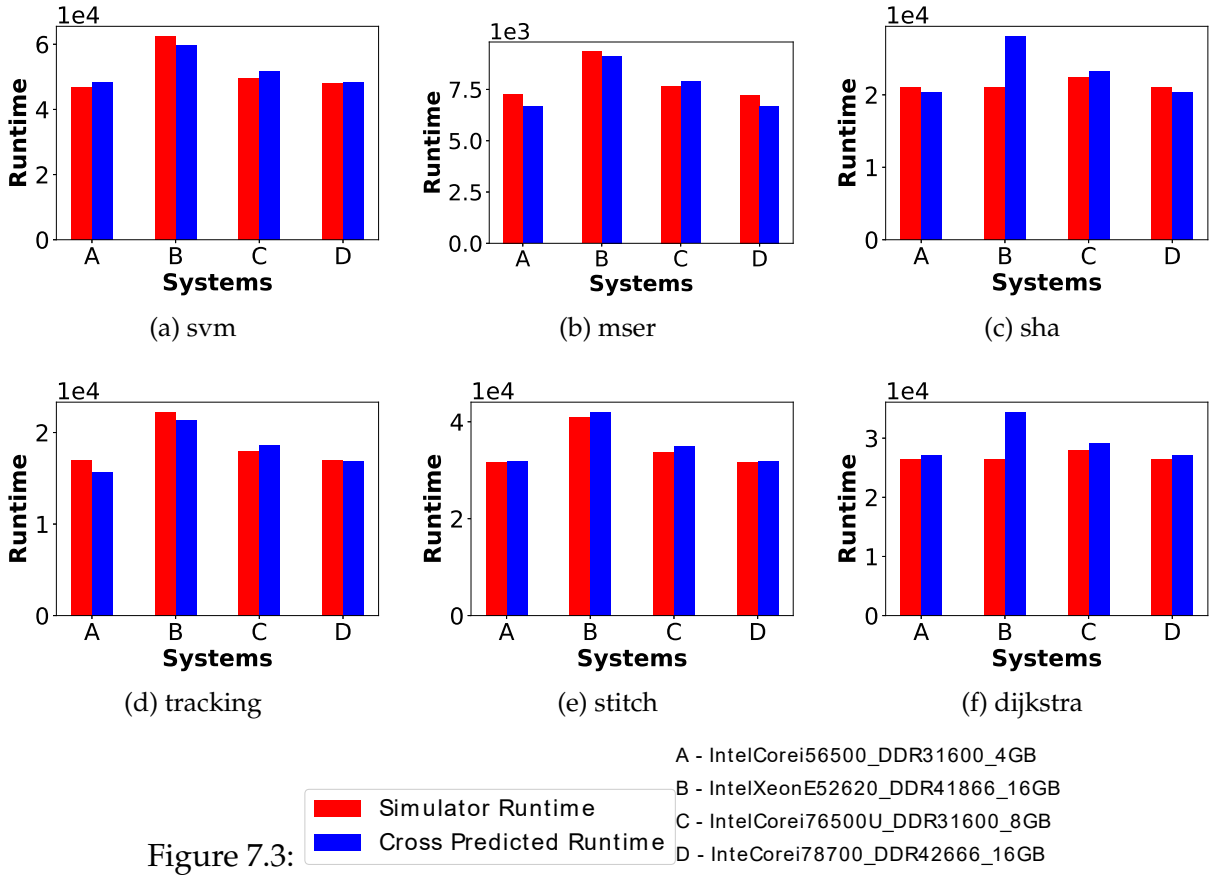


Figure 7.3: Simulator Runtime vs Cross Performance Prediction Runtime

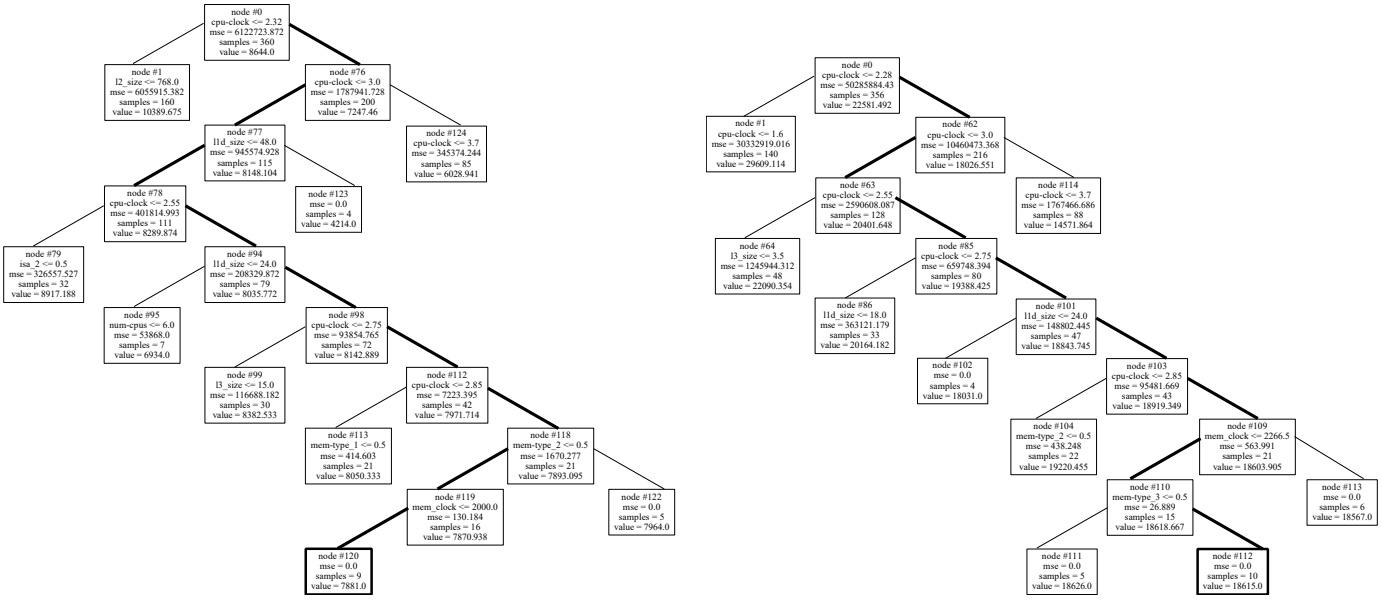
In the training phase, the decision tree regression model develops a binary tree with boolean logic defining a set of rules. Each index node of a tree has a rule (condition) with an associated threshold value with hardware features. The leaf node of this binary tree has the target value runtime. In the prediction phase, the index nodes' rules guide the path for a new set of hardware features to reach the leaf node determining the predicted runtime. To analyze how the decision tree

Table 7.3: True/Actual and Predicted Performance for Simulated Systems with Percentage Error

Algo	A			B			C			D		
	True	**Pred**	**PE**	**True**	**Pred**	**PE**	**True**	**Pred**	**PE**	**True**	**Pred**	**PE**
mser	7264	6670	8.18	9344	9130	2.29	7652	7881	2.99	7228	6707	7.21
svm	46909	48268	2.9	62389	59723	4.27	49703	51641	3.90	48000	48268	0.56
tracking	16999	15640	7.99	22209	21380	3.73	18019	18615	3.3	16965	16886	0.47
stitch	31758	31812	0.17	40884	41949	2.6	33690	34927	3.67	31617	31812	0.62
dijkstra	26446	27210	2.89	26446	34370	29.96	27951	29109	4.14	26431	27210	2.95
sha	21109	20350	3.6	21109	28053	32.9	22370	23235	3.87	21063	20350	3.38

predicts a particular runtime *yspred* for a given application on each of the four physical systems, we extracted the binary tree rules built by the trained decision tree regression model for each application. To extract these rules, we utilized the export_graphviz() function from scikit-learn, which generates a DOT file from each application's trained decision tree model. We then plotted the tree with index and leaf nodes using an open-source Graphviz tool. Figure 7.4 shows the binary trees built from the decision tree models for mser and tracking applications for system C as an example.

In the prediction phase, we took feature values from system C from table 7.2 and traverse through the index node either going to the left side when the rule condition is true or going to the right side when the rule's condition is false. In the case of mser, root node #0 with condition cpu-clock<=2.28 is false with System C having cpu-clock of 3 GHz, so we take the right branch. Following the sequence of conditions, we reach the leaf node with a runtime value of 7881, which is the predicted value shown in table 7.3 for mser on system C. Similarly, for tracking following the rules at index nodes of the binary tree, we reach the leaf with a runtime value of 18615, which is the predicted value for tracking on system C shown in table 7.3. The thick lines and thick leaf node border indicate the path that the prediction phase took for a given set of hardware features of systems C in these examples in figure 7.4.



(a) For mser system C

(b) For tracking system C

Figure 7.4: Decision Tree Rules Determining Predicted Runtime

7.5.2.2 Cross Performance Prediction Accuracy with and without Scaling

To demonstrate the accuracy of cross performance prediction, we compared actual runtimes from physical systems yp to cross predicted runtimes $yspred$ predicted from the learned model trained only on simulated systems performance dataset shown as a red and black bar in figure 7.5. We observed the mean error between cross predicted runtimes $yspred$ and actual runtimes yp of server systems B and D are 36.97% and 34.82% whereas for general-purpose systems A and C the mean errors are 18.82% and 21.46%. The higher errors in server systems are because gem5 simulated systems represent a general class of systems resulting in a larger gap in design between the simulated systems and physical server systems.

We observed a significant difference between physical systems' actual runtime yp and simulated physical systems' actual runtime ysp, called "Scaling Factor." We applied a scaling factor of 35% for compute-bound applications and 10% for memory-bound applications as determined from our previous work [47] to the unscaled cross predicted runtime $yspred$. We plotted unscaled and scaled cross predicted runtimes with respect to physical systems' actual runtimes in figure 7.5. We confirm from the figure that unscaled errors are much higher compared to the scaled errors. We have achieved a reduction using scaling factor for each appli-

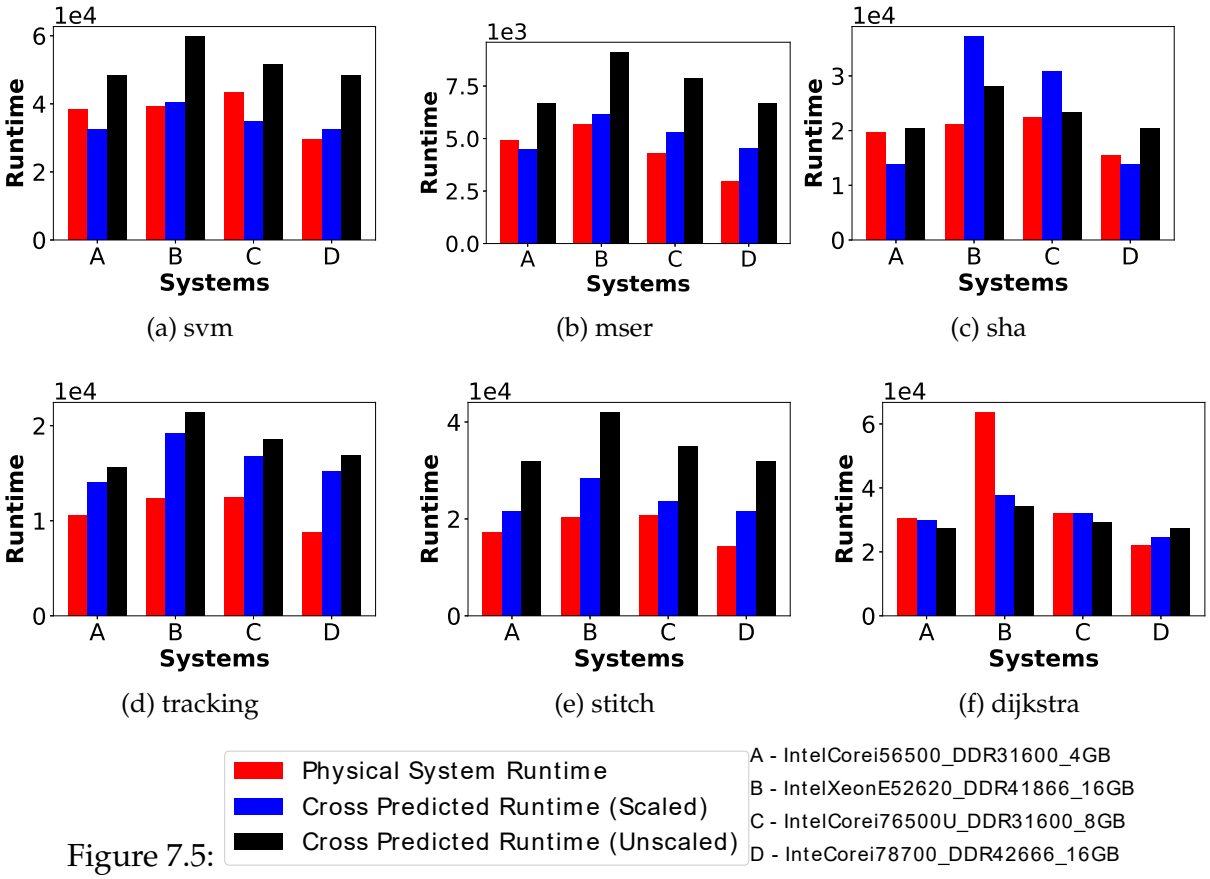


Figure 7.5: Physical System Runtime vs Cross Performance Prediction Runtime in Scaling Model

cation within the range of 10% to 25% for general-purpose systems except for sha and tracking application. The memory-bound tracking application has a higher error because it also depends on processor features; therefore, the scaling factor is slightly higher than 10% applied to it is cross predicted runtime. Similarly, the compute-bound sha application depends on memory features resulting in slightly different scaling factors than 35%.

7.6 Cross Performance Prediction with Transfer Learning

In this section, we discuss the experiments carried out for cross performance prediction with transfer learning. First, we provide experimental details about the construction of performance datasets using various schemes of samples selection

from simulated systems and source physical systems forming several instances of training datasets. We then show cross predictions for the same target physical system from the different training datasets. We analyze the predicted runtime for each target physical system to determine a source physical system (used for training) that provides the highest prediction accuracy for a given target physical system.

7.6.1 Experimental Details

We already built a simulated systems performance dataset $[Xs, ys]$ for each application with simulated systems hardware features and the application's actual runtime by executing an application on all 475 gem5 simulated systems. Similarly, we also built an application-specific performance dataset $[Xp, yp]$ for four physical systems by collecting hardware feature values using dmidecode utility and actual runtime by application's execution. We built a training dataset by combining simulated systems performance datasets and a source physical system performance dataset to apply transfer learning. The machine learning model, decision tree, is then trained using the training set to predict the runtime for the target physical system with different physical systems used as source and target.

We developed an algorithm 1 shown below for cross performance prediction with transfer learning. First, we selected m samples $PD_s = [Xs_m, ys_m]$ for the training phase from simulated systems performance dataset $[Xs, ys]$ with values of m varied to 10%, 50%, 90% and 100% of samples. Let the target physical systems set be $T = \{A, B, C, D\}$ and the selected target physical system be system t. We then selected a target physical system t's hardware features from performance dataset $PD_{pt} = [Xp_t]$ for cross performance prediction. We chose source physical systems set $S = \{A, B, C, D\} - t$ to have all the systems in the set except the target system. For example, if we use the system C as a target system, then the source systems set has systems A, B, and D. Therefore, a performance dataset of each source system s from set S for the same target physical system t is represented as $PD_{ps} = [Xp_s, yp_s]$. To reduce the effect of non-determinism on the runtime of the physical systems, we executed each application 40 times on each physical

system. We used the mean of all 40 runtime values as the actual runtime yp_t for the target physical system. On the other hand, to keep the larger number of samples in the training dataset from source physical systems, we used all 40 runtime value samples for source physical systems. For training phase, we merged the simulated systems performance dataset PD_s and source physical system dataset PD_{ps} into $PD_{training}$. We then trained the decision tree model on training dataset $PD_{training}$ called the "Learned Model." Finally, we provided the target physical system's hardware features to the learned model to predict a target physical system's performance yp_t for cross performance prediction.

Algorithm 1: Cross Performance Prediction using Transfer Learning

```
Let Simulated Systems Performance Dataset be [Xs, ys];
Let Physical Systems Performance Dataset be [Xp, yp];
Let All Physical Systems Set be {A, B, C, D};
Simulated Systems Dataset Sample Selection Percentage Set
  M = {10%, 50%, 90%, 100%};
for m ∈ M do
    Simulated Systems Performance Dataset for Training
      PD_s = [Xs_m, ys_m];
    Target Physical Systems Set T = {A, B, C, D};
    for t ∈ T do
        Target Physical Systems Performance Dataset for Cross Prediction
          PD_pt = [Xp_t];
        Source Physical Systems Set for Each Target System
          S = {A, B, C, D} − t;
        for s ∈ S do
            Source Physical Performance Dataset for Training
              PD_ps = [Xp_s, yp_s];
            Merge Simulated Systems and Source Physical Systems
              Performance Dataset for Training PD_training = PD_s + PD_ps;
            Train Decision Tree Machine Learning Model ("Learned
              Model") using Training Dataset dtr = dtr.fit(PD_training);
            Cross Performance Prediction for Target Physical Systems
              Using Learned Model yp_t = dtr.predict(PD_pt);
        end
    end
end
```

7.6.2 Results

We displayed prediction errors for each target in figure 7.6 with different source systems used for cross performance prediction in transfer learning model. We have three significant observations from the plot. The first observation is that the prediction errors for general-purpose systems A and C are lower in most cases than server systems B and D. Second observation is regarding the prediction accuracy of each target using different source systems from the plot itemized below.

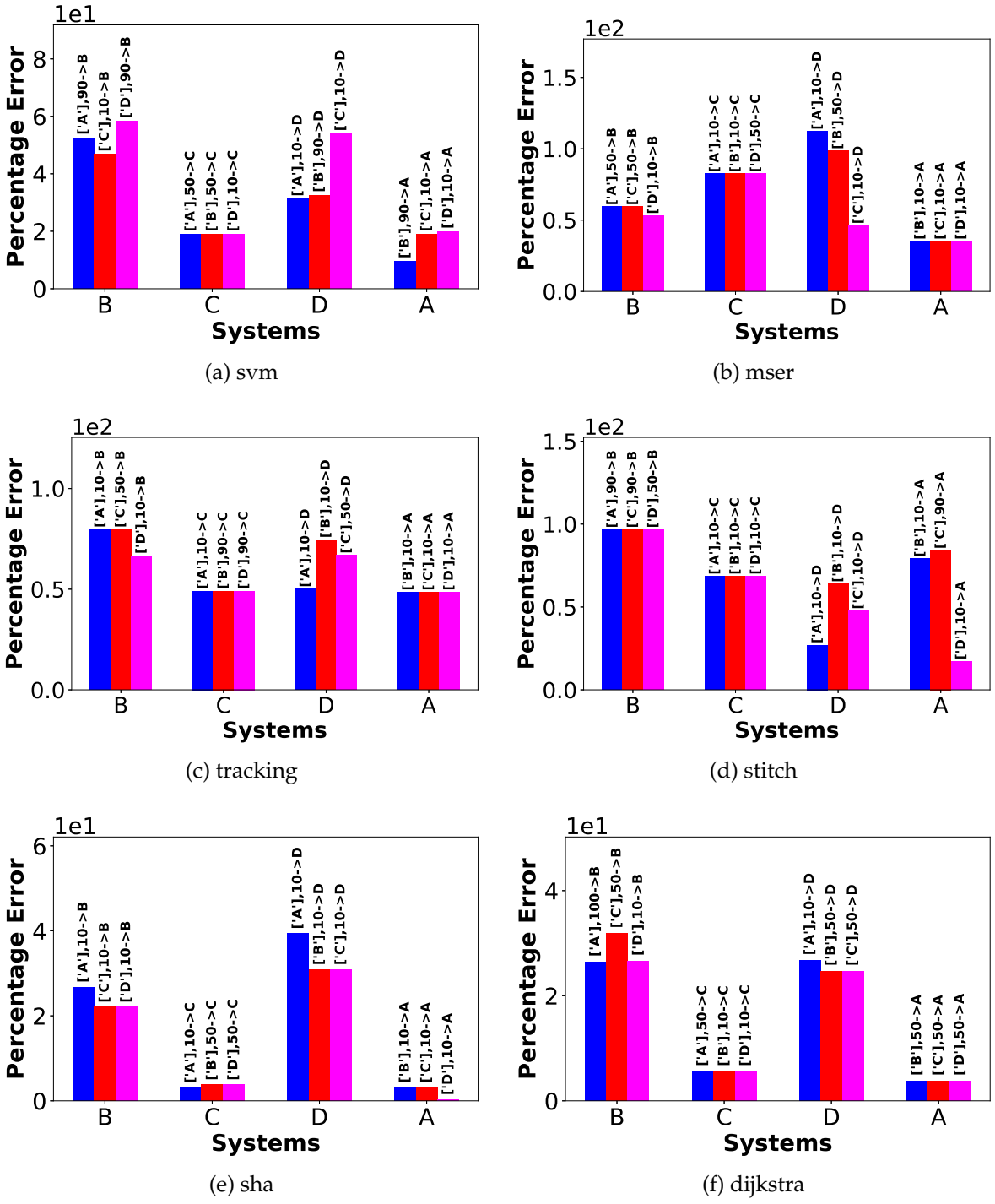
- Target system B has equal or lower accuracy for all applications except svm for cross prediction using source system D. This is because system B has a vastly different cpu-clock speed than the other three systems. However, system D, being a server system, they have other similar hardware features resulting in lower error when system D is used as a source system.

- For target system C, source system A has a lower or equal error for all applications because both are general-purpose systems with similar features.

- System D when used as a source physical system provided higher prediction accuracy for target system A except svm. We believe this is because a vital feature cpu-clock between systems A and D is the same, and cpu-clock having a higher correlation with runtime. Source system C also has good accuracy except for stitch for target system C since system A and system C are general-purpose systems.

- For target system D, we cannot identify a single source system with higher prediction accuracy, but system A or C has higher prediction accuracy than system B. This is because the important feature cpu-clock for A and C has a similar value as the cpu-clock of system D.

Finally, the third observation is that when comparing the prediction accuracy of different training datasets for each target, we notice that the training dataset with more than 50% of simulated systems' dataset samples has higher errors. This higher error is because using more than 50% samples from simulated systems performance dataset generates noise in the training performance dataset created by



(a) svm

(b) mser

(c) tracking

(d) stitch

(e) sha

(f) dijkstra

Figure 7.6: Prediction from each target system from different source systems in Transfer Learning Model

merging a mix of performance data from simulated systems and source physical system.

7.7 Prediction Accuracy: Scaling vs Transfer Learning

In this section, we compared performance prediction errors between cross performance prediction carried out using transfer learning in section 7.6 versus cross performance prediction with scaling in section 7.5. For comparison, we selected source systems with the lowest percentage error for each target system from cross performance prediction model with transfer learning. Figure 7.7 shows the percentage error for each application and each target using both the scaling and transfer learning techniques.

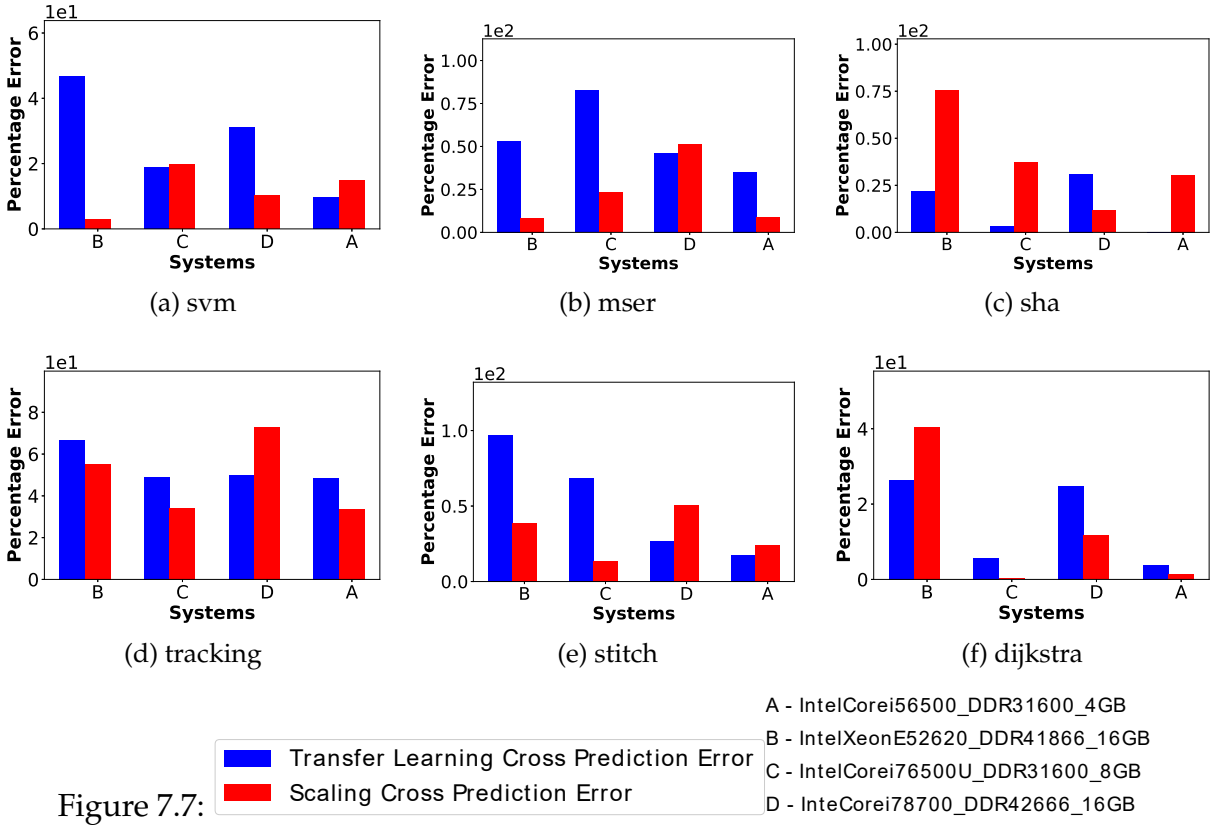


Figure 7.7: Comparison of Prediction Error for Transfer Learning vs Scaling

One important observation is that the target system's prediction accuracy for most of the applications except sha of cross performance prediction model with scaling is higher than cross performance prediction with transfer learning. We believe that this is because, with the help of the scaling factor, the scaling model is more effective in matching the predicted value with the actual runtime of target physical systems to bridge the gap in design between gem5 simulated systems

and physical systems. Furthermore, lower prediction accuracy in the transfer learning model is due to the use of a smaller number of samples from source physical systems in the training dataset making the transfer learning model less effective to bridge the design gap of simulated systems and physical systems. We believe that by having runtime collected from a larger number of physical systems with different hardware features used as source systems, the prediction accuracy of the transfer learning model can be improved, which we will consider in our future work.

7.8 Summary

This chapter implemented two cross performance prediction models, one with scaling and the other with transfer learning techniques to predict physical systems performance from the performance dataset of simulated systems. In cross performance prediction with scaling, we first predicted the physical system's performance using their hardware features and then applied the scaling factor decided from our previous work to get the actual predicted runtime. For cross performance prediction model with transfer learning, we combined the performance data from simulated systems and a source physical system to predict the target physical system's performance. We utilized the source physical system dataset along with a simulated systems dataset for incremental learning or providing a hint to the decision tree, a machine learning model.

We validated our models using six benchmark applications drawn from SD-VBS and MiBench suite, each with different computations and data access patterns representing separate application domains. We made the following conclusions from our work.

- Our cross performance prediction model with scaling factor provided a mean of 10-25% errors for general-purpose systems after applying scaling factor except for tracking and sha. Some dependence on computation caused a higher error in the memory-bound tracking application. In contrast, for compute-bound sha, the higher error resulted from some dependence on

memory features.

- We also developed a method to extract the set of rules in the form of a binary tree built during the decision tree regression model's training phase. This set of rules dictates the path traversal during the prediction phase by evaluating each condition at the index nodes to reach the predicted runtime value at the leaf node.

- For our cross performance prediction model with transfer learning, we used four physical systems. We selected each of the four systems as a target system and a different physical system as a source system from the set. We trained the decision tree regression, a machine learning model, using a source physical system and simulated systems combined performance dataset to predict the selected target's performance. We demonstrated that system C prediction accuracy is highest when system A is used as a source, both being a general-purpose system. Similarly, system D as a source has higher accuracy for system B as a target since both are server-like systems.

- Finally, we compared the prediction accuracy of both our cross performance prediction models. Our experimental results showed that the model with scaling outperforms the model with transfer learning for all applications except sha. We believe compute-bound sha has a higher error for scaling than the transfer learning model because the scaling factor requires adjustment depending on the dependence on memory features.

Our cross performance prediction model can be leveraged even for emerging hardware such as GPU. One can build simulated GPUs in a simulator such as GPGPU-Sim to apply our cross performance prediction model with scaling or transfer learning to predict the physical GPUs' performance for GPU benchmark applications. Similarly, these models can also be used for networked homogeneous systems if good simulators are available for simulating the networked systems. We would like to expand our cross prediction models research for heterogeneous and networked systems areas in the future.

CHAPTER 8

Summary, Conclusions and Future Work

In this section, first, we summarize our contributions and conclusions from the work performed in this dissertation. We then outline the tasks to guide the direction of the future work.

8.1 Summary and Conclusions

This thesis presented cross prediction models to predict the performance and power of physical computer systems from the machine learning model trained on the gem5 simulated systems' dataset. The cross prediction models are implemented using two innovative techniques transfer learning and scaling using machine learning algorithms. The thesis provided detailed results from both the cross prediction model for compute-intensive and data-intensive benchmark applications. The thesis work confirmed that accurate predictions from the cross prediction models can be used to address the problem of selection of physical computer systems without the need for executing the application on the actual computer system.

To build a foundation for our cross prediction model, we first established that a strong relationship exists between the hardware features of computer systems and the performance (runtime) of a given application executed on the respective computer systems. To capture this relationship in a model, we train machine learning algorithms from hardware features and performance and predict the performance given the hardware features of systems unseen during training. Experiment results show that even with identical hardware features in simulated and physical

systems, performance is vastly dissimilar due to the difference in design. Furthermore, we have shown that the relationship between hardware features and performance is globally non-linear resulting in higher prediction accuracy for non-linear machine learning models such as tree-based models.

It is important to use the machine learning algorithm that provides higher accuracy for our cross prediction models. Therefore, we investigated 14 univariate and multivariate machine learning algorithms. We have shown that univariate algorithms such as linear regression, gaussian process regression has lower prediction accuracy than multivariate algorithms such as tree-based and neural network algorithms for predictions of performance and power on simulated as well as physical systems. Experimental results show that models built from tree-based algorithms outperform all other univariate and multivariate algorithms. We believe that this is because a decision tree is a binary tree-based algorithm that effectively captures the non-linear relationship between hardware features and performance or power. The binary tree of the decision tree algorithm consists of non-leaf nodes with rules and leaf nodes with prediction values built during the training phase forming a boolean logic. During the prediction phase, the rules from non-leaf nodes are evaluated to reach the leaf node for prediction. We have shown that we can extract the rules from the binary tree of the trained decision tree algorithm to understand the boolean logic for prediction.

We have investigated the prediction accuracy of the tree-based machine learning model for different application types such as computationally-intensive and data-intensive applications for simulated systems and physical systems. We found that due to the deterministic nature of simulated systems prediction accuracy of simulated systems is much higher than non-deterministic physical systems when training and predictions are performed on the same dataset. We have also found that the computationally intensive applications have higher variations in runtime on physical systems resulting in higher prediction errors than data-intensive applications whereas the accuracy for power is similar in both types of applications. However, on simulated systems accuracy of computationally intensive applications is higher because of the use of the same processor model for all the simulated

systems built into the simulator.

Selecting physical system(s) for an application without having access to execute the applications on a plethora of physical systems with diversity in processor, cache and memory features makes the problem hard. We provide the solution to this problem by implementing a machine learning-based cross prediction model that effectively predicts the performance and power of physical systems from the systems built with the gem5 simulator. To speed up the construction of systems in the gem5 simulator, we leveraged the emulation mode of gem5 resulting in a wider gap between the performance and power collected from the gem5 simulated system with identical features to that of the physical system. Furthermore, experimental results revealed that the performance (runtime) gap is larger for compute-intensive applications compared to data-intensive applications due to the design of processors in simulated systems is much different than physical systems. Our "Cross Performance and Power Model with Scaling" provides the solution to bridge the gap by determining application-specific scaling factors, consisting of two components major factor and minor factor. The major factor is determined by identifying the mean difference of performance and power between the physical systems and gem5 simulated systems built using identical hardware features of the physical systems. The minor factor is derived from the Pearson correlation coefficient between hardware features and application-specific performance or power. We have shown that by accurate performance and power predictions from our scaling model, physical systems can be selected without executing the application on them eliminating the need to have access to the physical systems.

We provide another solution to the computer system(s) selection problem using a transfer learning technique called "Cross Performance and Power Prediction Model using Transfer Learning." Unlike the scaling model, the transfer learning model does not require determining the scaling factor, however, it assumes that access to some physical systems is available. Our transfer learning model uses the notion of a source physical system that is accessible, however, the target physical system is inaccessible requiring prediction. We have shown that a

machine learning-based transfer learning model trained from performance and power datasets of simulated systems plus a source physical system provides a reasonable prediction for the target physical system. Furthermore, experimental results reveal that the transfer learning model trained from a source physical system having hardware features close to that of the target physical systems provide higher prediction accuracy. The target systems predictions from the cross prediction model with transfer learning can be used to select computer system(s) even when access to target systems is unavailable.

8.2 Future Work Direction

In this section, we articulate the direction for cross prediction models to improve accuracy, adapt to applications with greater complexity and explore additional system architectures.

8.2.1 Improve Model Accuracy

There are two issues observed for model accuracy that requires improvement. First, experiment results from both cross prediction models show that the server-like systems have higher prediction errors. Our understanding from the result is that gem5 simulated systems are approximate to general-purpose systems such as laptops or desktops, hence, the scaling factor calculation is appropriate for these systems. However, there is a need to calculate the scaling factor specific to the server-like systems by considering a large number of server systems to calculate the major factor, a factor measuring the difference between physical server systems to the gem5 simulated systems. In the case of the transfer learning model, the prediction accuracy depends on the similarity between hardware features from the source and target physical system, therefore, we can improve the accuracy of the transfer learning model by having access to several server systems to be used as source systems with hardware features as close as possible to the target server systems. The second issue observed from the results is that the cross prediction model with transfer learning has errors higher than 50% on average. We

know that the prediction accuracy of the machine learning model is improved by having a training dataset representing a large number of physical systems samples hardware features close to that of the target system. Therefore, we believe by training the transfer learning model with a large number of physical systems samples having disparate hardware features as source systems should improve the accuracy.

8.2.2 Adapting to Complex Application

Although our cross prediction models have been implemented considering applications from a different domain with distinct computation and data access patterns, the model implementation provides the solution for single-threaded applications. However, high-performance computing systems (HPC) with multiple cores have been in use for a couple of decades to utilize the parallelized (multithreaded) versions of the applications for improving the performance of an application. In the future, we plan to investigate the possibility of enhancing cross prediction models for multithreaded and multiprocessing applications. To implement a cross prediction model for scaling for parallel applications, the main challenge is to determine different scaling factors depending on the number of threads or processes used by the application in addition to considering the actual number of cores of the systems. However, the cross prediction model with transfer learning does not have this requirement, hence, we believe it is better suited for the cross prediction of parallelized applications.

8.2.3 Explore Systems Architectures

In this thesis, our cross prediction model implementation is focused on predicting the performance and power consumption of physical systems with Intel processors having an x86-based instruction set. While computers with x86-based instruction sets are widely used, due to the proliferation of the internet of things (IoT) and edge computing, embedded systems are at the forefront of computing utilizing primarily ARM-based instruction sets. Therefore, a possible future work

is to perform cross prediction on embedded systems built using ARM instruction set. The gem5 simulator used for building simulated systems does support the ARM instruction set, therefore, we can use the same simulator for this work. For the cross prediction model, we have measured power consumption on physical systems using Intel's RAPL with the PAPI toolset. The main challenge for ARM-based embedded systems will be to develop a process for System-On-Chip (SoC) power measurement due to the utilization of different SoCs in disparate embedded systems. Additionally, the scaling model will require determining a new scaling factor for embedded systems with ARM-based processors.

References

[1] PICSAR, "Application of the roofline performance model to PICSAR," 2017.

[2] A. Ilic, F. Pratas, and L. Sousa, "Cache-aware Roofline model: Upgrading the loft," 2014.

[3] X. Zheng, L. K. John, and A. Gerstlauer, "Accurate phase-level cross-platform power and performance estimation," in *Proceedings - Design Automation Conference*, vol. 05-09-June, Institute of Electrical and Electronics Engineers Inc., jun 2016.

[4] Y. Kim, P. Mercati, A. More, E. Shriver, and T. Rosin, "P4: Phase-Based Power/Performance Prediction of Heterogeneous Systems via Neural Networks," in *2017 IEEE/ACM International Conference on Computer-Aided Design (ICCAD) digest of technical papers November 13-16, 2017, Irvine Marriot, Irvine CA*, (Irvine Marriot, Irvine CA), 2017.

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

[6] B. Li, L. Peng, and B. Ramadass, "Accurate and efficient processor performance prediction via regression tree based modeling," *Journal of Systems Architecture*, vol. 55, pp. 457–467, oct 2009.

[7] B. Ozisikyilmaz, G. Memik, and A. Choudhary, "Machine learning models to predict performance of computer system design alternatives," in *Proceedings of the International Conference on Parallel Processing*, pp. 495–502, 2008.

[8] B. K. Reddy, M. J. Walker, D. Balsamo, S. Diestelhorst, B. M. Al-Hashimi, and G. V. Merrett, "Empirical CPU power modelling and estimation in the gem5 simulator," in *2017 27th International Symposium on Power and Timing Modeling, Optimization and Simulation, PATMOS 2017*, vol. 2017-Janua, pp. 1–8, Institute of Electrical and Electronics Engineers Inc., nov 2017.

[9] A. Butko, F. Bruguier, A. Gamatié, G. Sassatelli, D. Novo, L. Torres, and M. Robert, "Full-System Simulation of big.LITTLE Multicore Architecture for Performance and Energy Exploration," in *Proceedings - IEEE 10th International Symposium on Embedded Multicore/Many-Core Systems-on-Chip, MCSoC 2016*, pp. 201–208, Institute of Electrical and Electronics Engineers Inc., dec 2016.

[10] A. Jooya, N. Dimopoulos, and A. Baniasadi, "Optimum power-performance GPU configuration prediction based on code attributes," in *Proceedings - 2017 International Conference on High Performance Computing and Simulation, HPCS 2017*, pp. 418–425, Institute of Electrical and Electronics Engineers Inc., sep 2017.

[11] Z. Yu, J. Wang, L. Eeckhout, and C. Xu, "QIG: Quantifying the Importance and Interaction of GPGPU Architecture Parameters," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 37, pp. 1211–1224, jun 2018.

[12] A. Kumar and B. Talawar, "Machine Learning Based Framework to Predict Performance Evaluation of On-Chip Networks," in *2018 Eleventh International Conference on Contemporary Computing (IC3) : 2-4 August 2018, Jaypee Institute of Information Technology, Noida, India*, (Noida, India), IEEE, 2018.

[13] G. A. Malazgirt and A. Yurdakul, "Prenaut: Design space exploration for embedded symmetric multiprocessing with various on-chip architectures," *Journal of Systems Architecture*, vol. 72, pp. 3–18, jan 2017.

[14] Y. Cheng, W. Chen, Z. Wang, and Y. Xiang, "Precise contention-aware performance prediction on virtualized multicore system," *Journal of Systems Architecture*, vol. 72, pp. 42–50, jan 2017.

[15] M. Amarıs, R. de Camargo, M. Dyab, A. Goldman, and D. Trystram, "A Comparison of GPU Execution Time Prediction using Machine Learning and Analytical Modeling," in *Proceedings, 2016 IEEE 15th International Symposium on Network Computing and Applications : 30 October-2 November 2016, Cambridge, MA, USA*, p. 399, IEEE, 2016.

[16] G. Inal and G. Kucuk, "Application of machine learning techniques on prediction of future processor performance," in *Proceedings - 2018 6th International Symposium on Computing and Networking Workshops, CANDARW 2018*, pp. 190–195, Institute of Electrical and Electronics Engineers Inc., dec 2018.

[17] L. Lopez, M. Guynn, and M. Lu, "Predicting Computer Performance Based on Hardware Configuration Using Multiple Neural Networks," in *Proceedings - 17th IEEE International Conference on Machine Learning and Applications, ICMLA 2018*, pp. 824–827, Institute of Electrical and Electronics Engineers Inc., jan 2019.

[18] J. L. Greathouse and G. H. Loh, "Machine learning for performance and power modeling of heterogeneous systems," in *IEEE/ACM International Conference on Computer-Aided Design, Digest of Technical Papers, ICCAD*, Institute of Electrical and Electronics Engineers Inc., nov 2018.

[19] R. Escobar and R. V. Boppana, "Performance Prediction of Parallel Applications Based on Small-Scale Executions," 2016.

[20] 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*, vol. 05-09-Dece, pp. 725–737, IEEE Computer Society, dec 2015.

[21] G. Mariani, A. Anghel, R. Jongerius, and G. Dittmann, "Predicting cloud performance for HPC applications before deployment," *Future Generation Computer Systems*, vol. 87, pp. 618–628, oct 2018.

[22] S. J. Pan and Q. Yang, "A survey on transfer learning," *IEEE Transactions on Knowledge and Data Engineering*, vol. 22, no. 10, pp. 1345–1359, 2010.

[23] 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, pp. 749–763, may 2020.

[24] 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.

[25] 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), pp. 9–18, IEEE Computer Society, dec 2019.

[26] P. Jamshidi, M. Velez, C. Kästner, N. Siegmund, and P. Kawthekar, "Transfer Learning for Improving Model Predictions in Highly Configurable Software," in *Proceedings - 2017 IEEE/ACM 12th International Symposium on Software Engineering for Adaptive and Self-Managing Systems, SEAMS 2017*, pp. 31–41, Institute of Electrical and Electronics Engineers Inc., jul 2017.

[27] J. Sifakis, "A vision for computer science-the system perspective," mar 2011.

[28] B. Chaudhury, A. Varma, Y. Keswani, Y. Bhatnagar, and S. Parikh, "Let's HPC: A web-based platform to aid parallel, distributed and high perfor-

mance computing education," *Journal of Parallel and Distributed Computing*, vol. 118, pp. 213–232, aug 2018.

[29] S. Williams, A. Waterman, and D. Patterson, "Roofline: An insightful visual performance model for multicore architectures," *Communications of the ACM*, vol. 52, pp. 65–76, apr 2009.

[30] G. Hager and G. Wellein, *Introduction to High Performance Computing for Scientists and Engineers*. Boca Raton, FL, USA: CRC Press, Inc., 1st ed., 2010.

[31] A. Butko, R. Garibotti, L. Ost, and G. Sassatelli, "Accuracy evaluation of GEM5 simulator system," in *ReCoSoC 2012 - 7th International Workshop on Reconfigurable and Communication-Centric Systems-on-Chip, Proceedings*, (York), University of York, 2012.

[32] 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," *ACM SIGARCH Computer Architecture News*, vol. 39, pp. 1–7, may 2011.

[33] 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*, vol. 65, pp. 367–381, feb 2016.

[34] M. Walker, S. Bischoff, S. Diestelhorst, G. Merrett, and B. Al-Hashimi, "Hardware-Validated CPU Performance and Energy Modelling," in *Proceedings - 2018 IEEE International Symposium on Performance Analysis of Systems and Software, ISPASS 2018*, pp. 44–53, Institute of Electrical and Electronics Engineers Inc., may 2018.

[35] T. Odajima, Y. Kodama, and M. Sato, "Power performance analysis of ARM scalable vector extension," in *21st IEEE Symposium on Low-Power and High-Speed Chips and Systems, COOL Chips 2018 - Proceedings*, pp. 1–3, Institute of Electrical and Electronics Engineers Inc., jun 2018.

[36] G. Yao, H. Yun, Z. P. Wu, R. Pellizzoni, M. Caccamo, and L. Sha, "Schedulability analysis for memory bandwidth regulated multicore real-time systems," *IEEE Transactions on Computers*, vol. 65, pp. 601–614, feb 2016.

[37] H. Wu, F. Liu, and R. B. Lee, "Cloud Server Benchmark Suite for Evaluating New Hardware Architectures," *IEEE Computer Architecture Letters*, vol. 16, pp. 14–17, jan 2017.

[38] S. Li, H. Ahn, R. Strong, J. Brockman, D. Tullsen, and N. Jouppi, "McPAT: An Integrated Power, Area, and Timing Modeling Framework for Multicore and Manycore Architectures," in *Proceedings of the 42nd Annual IEEE/ACM International Symposium on Microarchitecture : December 12-16, 2009, New York, New York, USA*, p. 585, Association for Computing Machinery, 2009.

[39] J. Cano-Cano, F. J. Andújar, F. J. Alfaro, and J. L. Sánchez, "Speeding up exascale interconnection network simulations with the VEF3 trace framework," *Journal of Parallel and Distributed Computing*, vol. 133, pp. 124–135, nov 2019.

[40] G. Wu, J. Greathousey, A. Lyashevskyy, N. Jayasenay, and D. Chiou, "GPGPU Performance and Power Estimation Using Machine Learning," in *2015 IEEE 21st International Symposium on High Performance Computer Architecture (HPCA) : date 7-11 Feb. 2015.*, IEEE, 2015.

[41] A. Mankodi, A. Bhatt, and B. Chaudhury, "Learning Based Performance Prediction of Algorithms on Disparate Computer Hardware Models.".

[42] A. Mankodi, A. Bhatt, B. Chaudhury, R. Kumar, and A. Amrutiya, "Evaluating Machine Learning Models for Disparate Computer Systems Performance Prediction," in *2020 IEEE International Conference on Electronics, Computing and Communication Technologies (CONECCT)*, pp. 1–6, IEEE, jul 2020.

[43] A. Mankodi, A. Bhatt, and B. Chaudhury, "Evaluation of neural network models for performance prediction of scientific applications," in *IEEE Region 10 Annual International Conference, Proceedings/TENCON*, vol. 2020-

Novem, pp. 426–431, Institute of Electrical and Electronics Engineers Inc., nov 2020.

[44] A. Mankodi, A. Bhatt, and B. Chaudhury, "Performance prediction from simulation systems to physical systems using machine learning with transfer learning and scaling," *Concurrency and Computation: Practice and Experience*, p. e6433, jun 2021.

[45] A. Mankodi, A. Bhatt, and B. Chaudhury, "Multivariate Performance and Power Prediction of Algorithms on Simulation-Based Hardware Models," in *2020 19th International Symposium on Parallel and Distributed Computing (ISPDC)*, pp. 150–157, IEEE, jul 2020.

[46] A. Mankodi, A. Bhatt, B. Chaudhury, R. Kumar, and A. Amrutiya, "Modeling Performance and Power on Disparate Platforms Using Transfer Learning with Machine Learning Models," in *Proceedings of CoMSO 2020 - Modeling, Simulation and Optimization*, pp. 231–246, Springer, 2021.

[47] A. Mankodi, A. Bhatt, and B. Chaudhury, "Performance Prediction of Physical Computer Systems Using Simulation-Based Hardware Models," in *2020 International Conference on High Performance Big Data and Intelligent Systems, HPBD and IS 2020*, (Beijing), pp. 1–5, Chinese Academy of Sciences, IEEE, may 2020.

[48] A. Mankodi, A. Bhatt, and B. Chaudhury, "Predicting physical computer systems performance and power from simulation systems using machine learning model," *Computing*, pp. 1–19, mar 2022.

[49] R. Kumar, A. Mankodi, A. Bhatt, B. Chaudhury, and A. Amrutiya, "Cross-Platform Performance Prediction with Transfer Learning using Machine Learning," in *2020 11th International Conference on Computing, Communication and Networking Technologies, ICCCNT 2020*, Institute of Electrical and Electronics Engineers Inc., jul 2020.

[50] 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, pp. 1488–1514, dec 2017.

[51] I. Baldini, S. J. Fink, and E. Altman, "Predicting GPU performance from CPU runs using machine learning," in *Proceedings - Symposium on Computer Architecture and High Performance Computing*, pp. 254–261, IEEE Computer Society, dec 2014.

[52] A. Evangelidis, D. Parker, and R. Bahsoon, "Performance modelling and verification of cloud-based auto-scaling policies," *Future Generation Computer Systems*, vol. 87, pp. 629–638, oct 2018.

[53] G. Lu, W. Zhang, H. He, and L. T. Yang, "Performance modeling for MPI applications with low overhead fine-grained profiling," *Future Generation Computer Systems*, vol. 90, pp. 317–326, jan 2019.

[54] J. Guo, A. Ma, Y. Yan, and B. Zhang, "Application performance prediction method based on cross-core performance interference on multi-core processor," *Microprocessors and Microsystems*, vol. 47, pp. 112–120, nov 2016.

[55] D. Nemirovsky, T. Arkose, N. Markovic, M. Nemirovsky, O. Unsal, and A. Cristal, "A Machine Learning Approach for Performance Prediction and Scheduling on Heterogeneous CPUs," in *Proceedings - 29th International Symposium on Computer Architecture and High Performance Computing, SBAC-PAD 2017*, pp. 121–128, Institute of Electrical and Electronics Engineers Inc., nov 2017.

[56] P. J. J. Kapil, V. Matthew, and J. Thazhuthaveetil, "Construction and Use of Linear Regression Models for Processor Performance Analysis," tech. rep., 2006.

[57] L. Breiman, J. H. Friedman, R. A. Olshen, and C. J. Stone, *Classification and regression trees*. The Wadsworth statistics/probability series, Monterey, CA: Wadsworth and Brooks/Cole Advanced Books and Software, 1984.

[58] 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.

[59] M. A. Heroux, D. W. Doerfler, P. S. Crozier, J. M. Willenbring, H. Carter Edwards, A. Williams, M. Rajan, E. R. Keiter, H. K. Thorn-Quist, and R. W. Numrich, "Improving Performance via Mini-applications," tech. rep., 2009.

[60] Gem5, "Gem5: O3CPU," 2012.

[61] K. Asanovic, R. Bodik, B. Christopher, C. Joseph, J. Gebis, P. Husbands, K. Keutzer, D. A. Patterson, W. Lester, P. John, S. Samuel, W. Williams, and K. A. Yelick, "The Landscape of Parallel Computing Research: A View from Berkeley," tech. rep., 2006.

[62] K. Asanovic, R. Bodik, J. Demmel, T. Keaveny, K. Keutzer, J. Kubiatowicz, N. Morgan, D. Patterson, K. Sen, J. Wawrzynek, D. Wessel, and K. Yelick, "A view of the parallel computing landscape," *Communications of the ACM*, vol. 52, pp. 56–67, oct 2009.

[63] 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 *2001 IEEE International Workshop on Workload Characterization, WWC 2001*, pp. 3–14, Institute of Electrical and Electronics Engineers Inc., 2001.

[64] Carey G, "Coding Categorical Variables," tech. rep., 2003.

[65] 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 É. Duchesnay, "Scikit-learn: Machine Learning in Python Gaël Varoquaux Bertrand Thirion Vincent Dubourg Alexandre Passos PEDREGOSA, VAROQUAUX, GRAMFORT ET

AL. Matthieu Perrot," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[66] T. Hoefler and R. Belli, "Scientific benchmarking of parallel computing systems: Twelve ways to tell the masses when reporting performance results," in *International Conference for High Performance Computing, Networking, Storage and Analysis, SC*, vol. 15-20-Nove, IEEE Computer Society, nov 2015.

[67] G. James, D. Witten, T. Hastie, and R. Tibshirani, *An Introduction to Statistical Learning: With Applications in R*. Springer Publishing Company, Incorporated, 2014.

[68] Intel Corporation, "Intel: Product Specification," 2019.

[69] W. contributors, "List of interface bit rates — Wikipedia, The Free Encyclopedia," 2019.

[70] 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.

[71] Awad Mariette, , and R. Khanna, "Support Vector Regression," in *Efficient Learning Machines: Theories, Concepts, and Applications for Engineers and System Designers*, pp. 67–80, Berkeley, CA: Apress, 2015.

[72] A. E. Hoerl and R. W. Kennard, "Ridge Regression: Biased Estimation for Nonorthogonal Problems," *Technometrics*, vol. 12, no. 1, pp. 55–67, 1970.

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

[74] C. E. Rasmussen, "Gaussian Processes in Machine Learning," in *Advanced Lectures on Machine Learning: ML Summer Schools 2003, Canberra, Australia, February 2 - 14, 2003, Tübingen, Germany, August 4 - 16, 2003, Revised Lectures* (Bousquet Olivier, , U. von Luxburg, Gunnar, and Rätsch, eds.), pp. 63–71, Berlin, Heidelberg: Springer Berlin Heidelberg, 2004.

[75] L. Breiman, "Random Forests," *Machine Learning*, vol. 45, no. 1, pp. 5–32, 2001.

[76] P. Geurts, D. Ernst, and L. Wehenkel, "Extremely randomized trees," *Machine Learning*, vol. 63, pp. 3–42, apr 2006.

[77] J. H. Friedman, "Stochastic gradient boosting," *Computational Statistics & Data Analysis*, vol. 38, pp. 367–378, feb 2002.

[78] 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.

[79] F. Chollet and Others, "Keras." https://keras.io, 2015.

[80] R. Kohavi, "A Study of Cross-Validation and Bootstrap for Accuracy Estimation and Model Selection," tech. rep., 1995.

[81] D. P. Kingma and J. Ba, "Adam: A Method for Stochastic Optimization," dec 2014.

[82] M. Pricopi, T. S. Muthukaruppan, V. Venkataramani, T. Mitra, and S. Vishin, "Power-performance modeling on asymmetric multi-cores," in *2013 International Conference on Compilers, Architecture and Synthesis for Embedded Systems (CASES)*, pp. 1–10, 2013.

[83] M. Rajat, , I. Banicescu, Srishti, Srivastava, Sherif, and Abdelwahed, "A Power-Aware Autonomic Approach for Performance Management of Scientific Applications in a Data Center Environment," in *Handbook on Data Centers* (Khan Samee U., , and A. Y. Zomaya, eds.), pp. 163–189, New York, NY: Springer New York, 2015.

[84] M. Aldossary, K. Djemame, I. Alzamil, A. Kostopoulos, A. Dimakis, and E. Agiatzidou, "Energy-aware cost prediction and pricing of virtual machines in cloud computing environments," *Future Generation Computer Systems*, vol. 93, pp. 442–459, apr 2019.

[85] S. Kawaguchi and T. Yachi, "Adaptive power efficiency control by computer power consumption prediction using performance counters," *IEEE Transactions on Industry Applications*, vol. 52, pp. 407–413, jan 2016.

[86] 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*, vol. 11, pp. 600–612, jun 2017.

[87] 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.

[88] D. Xu, Y. Shi, I. W. Tsang, Y.-S. Ong, C. Gong, and X. Shen, "Survey on Multi-Output Learning," *IEEE Transactions on Neural Networks and Learning Systems*, pp. 1–21, nov 2019.

[89] L. Yang, L. Wang, X. Zhang, and D. L. Wang, "An approach to build cycle accurate full system VLIW simulation platform," *Simulation Modelling Practice and Theory*, vol. 67, pp. 14–28, sep 2016.

[90] L. Breiman, J. H. Friedman, R. A. Olshen, and C. J. Stone, *Classification and regression trees*. 2017.

[91] P. Ranganathan, "Recipe for efficiency: Principles of power-aware computing," *Communications of the ACM*, vol. 53, pp. 60–67, apr 2010.

[92] D. Xu, Y. Shi, I. W. Tsang, Y.-S. Ong, C. Gong, and X. Shen, "A Survey on Multi-output Learning," jan 2019.

[93] S. Li, J. H. Ahn, R. D. Strong, J. B. Brockman, D. M. Tullsen, and N. P. Jouppi, "The McPAT framework for multicore and manycore architectures: Simultaneously modeling power, area, and timing," *Transactions on Architecture and Code Optimization*, vol. 10, pp. 1–29, apr 2013.

[94] Wikipedia contributors, "Pearson correlation coefficient — Wikipedia, The Free Encyclopedia."

[95] H. Terpstra Dan and Jagode, Y. Haihang, and D. Jack, "Collecting Performance Data with PAPI-C," in *Tools for High Performance Computing 2009* (M. M. Müller Matthias S. and Resch, S. Alexander, and N. W. E, eds.), (Berlin, Heidelberg), pp. 157–173, Springer Berlin Heidelberg, 2010.

[96] Terpstra, "PAPITopics: Accessing the Intel RAPL Registers," 2013.

[97] W.-Y. Loh, "Classification and regression trees," *WIREs Data Mining and Knowledge Discovery*, vol. 1, no. 1, pp. 14–23, 2011.

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

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

[100] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding," oct 2018.

[101] W. contributors, "Decision tree learning — Wikipedia, The Free Encyclopedia," 2020.

[102] X. Wu, V. Kumar, Q. J. Ross, J. Ghosh, Q. Yang, H. Motoda, G. J. McLachlan, A. Ng, B. Liu, P. S. Yu, Z. H. Zhou, M. Steinbach, D. J. Hand, and D. Steinberg, "Top 10 algorithms in data mining," *Knowledge and Information Systems*, vol. 14, pp. 1–37, jan 2008.

[103] W. Alkohlani and J. Cook, "Towards performance predictive application-dependent workload characterization," in *Proceedings - 2012 SC Companion: High Performance Computing, Networking Storage and Analysis, SCC 2012*, pp. 426–436, 2012.

[104] F. Bonomi, R. Milito, P. Natarajan, and J. Zhu, "Fog computing: A platform for internet of things and analytics," *Studies in Computational Intelligence*, vol. 546, pp. 169–186, 2014.

[105] G. Carey, "Psychology 5741 (Neuroscience) Logistic Regression," *The American Statistician*, vol. 5741, no. 5, pp. 1–15, 2003.

[106] T. Chen, Q. Guo, O. Temam, Y. Wu, Y. Bao, Z. Xu, and Y. Chen, "Statistical performance comparisons of computers," *IEEE Transactions on Computers*, vol. 64, pp. 1442–1455, may 2015.

[107] U. Gupta, M. Babu, R. Ayoub, M. Kishinevsky, F. Paterna, S. Gumussoy, and U. Y. Ogras, "An online learning methodology for performance modeling of graphics processors," *IEEE Transactions on Computers*, vol. 67, pp. 1677–1691, dec 2018.

[108] P. Jamshidi, N. Siegmund, M. Velez, C. Kastner, A. Patel, and Y. Agarwal, "Transfer learning for performance modeling of configurable systems: An exploratory analysis," in *ASE 2017 - Proceedings of the 32nd IEEE/ACM International Conference on Automated Software Engineering*, pp. 497–508, Institute of Electrical and Electronics Engineers Inc., nov 2017.

[109] W. Lee, Y. Kim, J. H. Ryoo, D. Sunwoo, A. Gerstlauer, and L. K. John, "PowerTrain: A learning-based calibration of McPAT power models," in *Proceedings of the International Symposium on Low Power Electronics and Design*, vol. 2015-Septe, pp. 189–194, Institute of Electrical and Electronics Engineers Inc., sep 2015.

[110] X. Li, X. Jiang, P. Garraghan, and Z. Wu, "Holistic energy and failure aware workload scheduling in Cloud datacenters," *Future Generation Computer Systems*, vol. 78, pp. 887–900, jan 2018.

[111] Z. Li, H. Wu, and S. He, "Timing Prediction for Dynamic Application Migration on Multi-core Embedded Systems," in *2018 IEEE 4th International Conference on Big Data Security on Cloud (BigDataSecurity), IEEE International Conference on High Performance and Smart Computing, (HPSC) and IEEE International Conference on Intelligent Data and Security (IDS)*, pp. 159–164, 2018.

[112] Z. Li, S. He, and L. Wang, "Prediction Based Run-Time Reconfiguration on Many-Core Embedded Systems," in *Proceedings - 2017 IEEE International*

Conference on Computational Science and Engineering and IEEE/IFIP International Conference on Embedded and Ubiquitous Computing, CSE and EUC 2017, vol. 2, pp. 140–146, Institute of Electrical and Electronics Engineers Inc., aug 2017.

[113] D. Loghin, L. Ramapantulu, O. Barbu, and Y. M. Teo, "A time–energy performance analysis of MapReduce on heterogeneous systems with GPUs," *Performance Evaluation*, vol. 91, pp. 255 – 269, 2015.

[114] Y. Lu, X. Wang, W. Zhang, H. Chen, L. Peng, and W. Zhao, "Performance Analysis of Multimedia Retrieval Workloads Running on Multicores," *IEEE Transactions on Parallel and Distributed Systems*, vol. 27, pp. 3323–3337, nov 2016.

[115] W. S. McCulloch and W. Pitts, "A logical calculus of the ideas immanent in nervous activity," *The bulletin of mathematical biophysics*, vol. 5, no. 4, pp. 115–133, 1943.

[116] P. Mejia-Alvarez, D. Moncada-Madero, H. Aydin, and A. Diaz-Ramirez, "Evaluation framework for energy-aware multiprocessor scheduling in real-Time systems," *Journal of Systems Architecture*, vol. 98, pp. 388–402, sep 2019.

[117] M. Pranzo and S. Mazumdar, "An analytical model for thread-core mapping for tiled CMPs," *Performance Evaluation*, vol. 134, p. 102003, 2019.

[118] J. Rico-Gallego, J. MartÃn, R. Manumachu, and A. Lastovetsky, "A Survey of Communication Performance Models for High-Performance Computing," *ACM Computing Surveys*, vol. 51, pp. 1–36, apr 2019.

[119] D. Rupanetti and H. Salamy, "Task allocation, migration and scheduling for energy-efficient real-time multiprocessor architectures," *Journal of Systems Architecture*, vol. 98, pp. 17–26, sep 2019.

[120] P. Singleton, "Performance Modelling - What, Why, When and How," *BT Technology Journal*, 2002.

[121] R. Vazquez, A. Gordon-Ross, and G. Stitt, "Machine Learning-based Prediction for Dynamic Architectural Optimizations," in *2019 10th International Green and Sustainable Computing Conference, IGSC 2019*, Institute of Electrical and Electronics Engineers Inc., oct 2019.

[122] Y. Wen, Z. Wang, Y. Zhang, J. Liu, B. Cao, and J. Chen, "Energy and cost aware scheduling with batch processing for instance-intensive IoT workflows in clouds," *Future Generation Computer Systems*, vol. 101, pp. 39–50, dec 2019.

[123] J. Zhou, J. Yan, K. Cao, Y. Tan, T. Wei, M. Chen, G. Zhang, X. Chen, and S. Hu, "Thermal-aware correlated two-level scheduling of real-time tasks with reduced processor energy on heterogeneous MPSoCs," *Journal of Systems Architecture*, vol. 82, pp. 1–11, jan 2018.

CHAPTER A

Glossary

Table A.1: Glossary

Acronym	Full Form
FSS	Full System Simulator
LR	Linear Regression
DTR	Decision Tree Regression
McPAT	Multicore Power, Area and Timing
PAPI	Performance Application Programming Interface
API	Application Programming Interface
RAPL	Running Average Power Limit
SD-VBS	San Diego Vision Benchmark Suite
MiBench	eMbedded Becnhmark
GPU	Graphical Processing Unit
OoO	out-of-order
ISA	Instruction-Set-Architecture
CPU	Central Processing Unit