PERFORMANCE AND POWER ESTIMATION USING DATAMINING THROUGH MACHINE LEARNING TECHNIQUES

M.Tech. Thesis

By

GAYATRI VIJAYAKUMAR

DISCIPLINE OF ELECTRICAL ENGINEERING

INDIAN INSTITUTE OF TECHNOLOGY INDORE

JUNE 2020

PERFORMANCE AND POWER ESTIMATION USING DATAMINING THROGH MACHINE LEARNING TECHNIQUES

A THESIS

Submitted in partial fulfillment of the
requirements for the award of the degree
of
Master of Technology

by

GAYATRI VIJAYAKUMAR

DISCIPLINE OF ELECTRICAL ENGINEERING
INDIAN INSTITUTE OF TECHNOLOGY INDORE
JUNE 2020

INDIAN INSTITUTE OF TECHNOLOGY INDORE

CANDIDATE'S DECLARATION

I hereby certify that the work which is being presented in the thesis entitled **PERFORMANCE AND POWER ESTIMATION USING DATAMINING THROGH MACHINE LEARNING TECHNIQUES** in the partial fulfillment of the requirements for the award of the degree of **MASTER OF TECHNOLOGY** and submitted in the **DISCIPLINE OF ELECTRICAL ENGINEERING, Indian Institute of Technology Indore**, is an authentic record of my own work carried out during the time period from July 2018 to June 2020 under the supervision of Dr. Abhinoy Kumar Singh, Inspire Faculty and Dr. Saptarshi Ghosh, Assistant Professor.

The matter presented in this thesis has not been submitted by me for the award of any other degree of this or any other institute.

19-06-2020

Signature of the student with date
GAYATRI VIJAYAKUMAR

This is to certify that the above statement made by the candidate is correct to the best of my/our knowledge.

22/06/2020

Signature of the Supervisor of
M.Tech. thesis (with date)
DR. ABHINOY KUMAR SINGH

Saptarshi Ghosh
22/06/2020.

Signature of the Supervisor of
M.Tech. thesis (with date)
DR. SAPTARSH GHOSH

GAYATRI VIJAYAKUMAR has successfully given his/her M.Tech. Oral Examination held on **22-06-2020**.

(Abhinoy Kr. Singh)
Saptarshi Ghosh

Signature(s) of Supervisor(s) of M.Tech. thesis (SAPTARSHI GHOSH)
Date: 22/06/2020

Convener, DPGC
Date: 22-06-2020

Somnath Dey 22/06/2020

Signature of PSPC Member #1
Date:

Signature of PSPCMember #1
Date: 22.06.2020

ACKNOWLEDGEMENTS

First and foremost, I would like to thank God Almighty for giving me the strength, knowledge, ability and opportunity to undertake this Thesis work and to persevere and proceed successfully.

I would like to express my sincere gratitude to my Thesis Supervisors, **Dr. Abhinoy Kumar Singh** and **Dr. Saptarshi Ghosh** for their constant support, encouragement and guidance throughout the Thesis work. I would also like extend my heartfelt gratitude to my PSPC members, **Dr. Ram Bilas Pachori** and **Dr. Somnath Dey** and CSP Program Coordinator, **Dr. Swaminathan R** for their valuable suggestions and feedback.

Furthermore, I would also like to thank all the faculty members and the staff at IIT Indore for their cooperation throughout my study and thesis work. I am thankful to the Discipline of Electrical Engineering of IIT Indore for providing all the facilities and resources required for the completion of this work.

I am extremely indebted to **Melwyn Scudder**, Engineering Manager, Intel Technology who has always been an exemplary and visionary mentor. He always found time off his busy schedule to address my doubts and guide me through the right path by his invaluable technical expertise.

I am also grateful to all my colleagues who were always a constant source of happiness and motivation for their unfailing support and continuous encouragement throughout my years of study. Finally, I express my profound gratitude to my family who always believed in me and who are my pillars of strength. This accomplishment would have been impossible without them.

Dedicated to my family

ABSTRACT

The development of fast and efficient processors is an inevitable requirement in today's automated world. New architectures that are robust and can handle complex real-time applications have to be developed. But how to find the optimum architecture and the hardware configurations in the shortest and accurate way is an open question. If the configurations must be changed after the prototype is created since it does not satisfy the customer requirements, then it will become a tedious and time-consuming process and the whole flow will need to get repeated again which is highly undesirable. Therefore, an early and exact determination of an efficient processor architecture is needed before the hardware development even starts.

Modern processing systems with heterogeneous components have numerous configuration and design options such as the number and types of cores, frequency, and memory bandwidth. Hardware architects are confronted with hundreds of design parameters which can be combined in many arbitrary ways to develop new architectures. Different hardware and software configuration parameters should be evaluated by estimating the power and performance corresponding to each set without the availability of the real hardware. This highlights the importance of rapid performance and power estimation mechanisms.

In this work, we propose a method to estimate power and performance of different workloads for a hardware configuration using machine learning techniques. After training the model with the data from previous generation processors, we will be able to predict the performance and power of the new generation processors with different hardware and workload features associated with that processor without having to run the simulation. This can help the architecture design team to select the best architectural configuration and build the design with optimal power and performance in a faster way to decrease turnaround times in the product lifecycle and increase the product goodness.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACRONYMS

SOC	System On Chip
PRQ	Production Release Qualification
ML	Machine Learning
IPC	Instructions Retired per cycle
BPU	Branch Prediction Unit
TLB	Translation Look-aside buffer
RMSE	Root Mean Square Error
EC	Event cost
AF	Activity Factor
EDA	Exploratory Data Analysis
URL	Uniform Resource Locator
HDFS	Hadoop Distributed File System

Chapter

1. Introduction

1.1 Power and Performance Estimation

Power and Performance estimation of processors is an important stage in the development of an SOC because this will eventually pave way to provide the best architecture possible to meet the demands of the customer [1]. Benchmark developers work on modelling the processor performance for a given workload and processor architects estimate the power and performance with respect to these workloads. Thus, while developing the architecture, hardware architects perform pathfinding with respect to performance and power, incorporates the changes to be made thereby developing the best architecture for production. Performance and Power estimation is an important and crucial stage in SOC development to meet the time-to-market constraints. So, developing a power and performance estimation model which is fast and accurate can be very useful to processor architects since it will help them to design and fine tune future processors. It can help them to understand how the variation in different knobs like frequency, cache size, etc. can affect power and performance. From this they can understand how a processor would behave when subjected to a particular workload which emulates a certain application or event.

1.2 Power and Performance Engineering

Performance improvement is a constant demand from customers and it's a responsibility to set and meet new performance and power targets with each new generation of processors. Performance improvement is as important as functional improvement since a new feature added by compromising performance is futile. Thus, always performance should be kept in mind while any architectural changes are made. In the product lifecycle we have Technology Readiness [2], Product Definition, Pre - Silicon Validation, Post Silicon Validation and Post PRQ. Technology Readiness includes setting product performance requirements, Workload modelling and Design Space Exploration using different methods like analytical models. Performance Requirements are set by various factors like Current product Performance, Market Research, Customer Feedback, Academic Research, Industry trends, Generational Requirements, Competitive Requirements, Landing zone requirements.

1.3 What are workloads?

Workloads or benchmarks are a means to measure how the processor performs in a customer environment. In fact, the workloads emulate the customer environments. Workload creation is in itself a whole different domain where lot of effort is put to create the workloads that best mimic the environments on which the processors run [3]. It helps us to know how well our processor will perform by measuring their power and performance before it actually reaches the customers and ranks them among their peers. So, finding a variety of workloads that together represent a large fraction

of customer environments is crucial to meet product performance objectives. Then we analyze how these workloads scale over a variety of parameters like frequency changes, core counts, cache size, etc.

1.4 Design Space Exploration

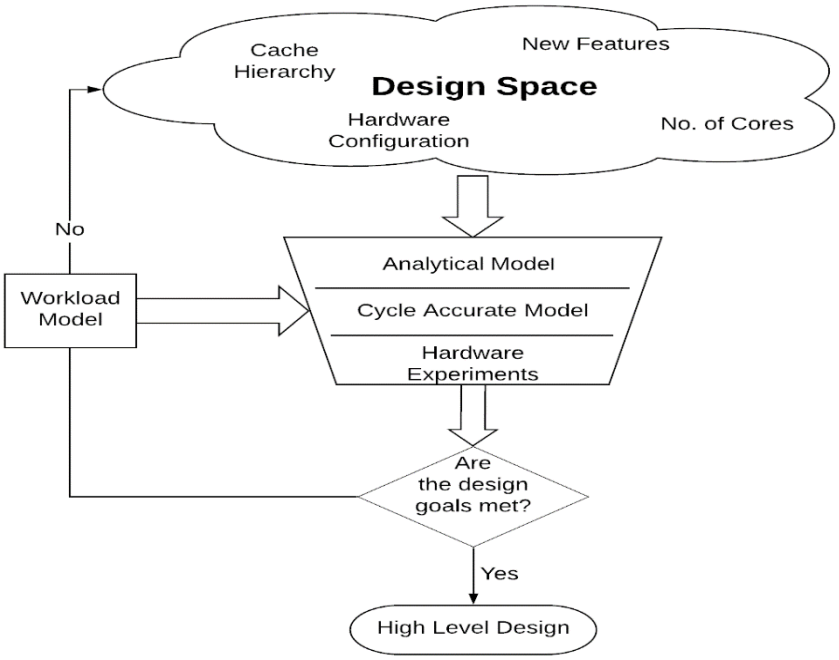

Figure 1. 1: Workflow in power and performance estimation

The workloads which emulate the customer environments are fed to different models like analytical, cycle accurate models or through hardware experiments by which the processor performance and power is calculated. If the design goals are not met then we go back and change different knobs like frequency, cache size, number of cores and other hardware configurations and estimate the power and performance gains using any of the models. This process is

repeated until the design goals are met and finally high-level design is formed. Figure 1.1 illustrates the overall workflow till the high-level design formation.

1.5 Artificial Intelligence

Artificial Intelligence is a branch which is growing and being used extensively to solve many problems in day-to-day lives and in the industries [4]. It is a way to enable machines to think like human brain. Machine learning is a subset of Artificial Intelligence where the system learns from the data given to it. It analyzes, understands and finds a certain pattern in the data. It can take decisions with minimum human intervention. The computer is trained to automate tasks which would otherwise be impossible or exhaustive for a human being. Deep Learning is a subset of machine learning that mimics the functioning of neurons in human body. It uses neural networks to analyze different patterns in the data. The depth of the model is decided by the number of layers in the model. Figure 1.2 describes how Artificial Intelligence, Machine Learning and Deep Learning are related.

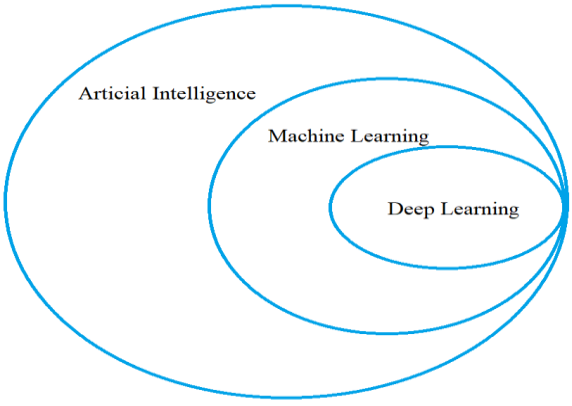

Figure 1. 2: Artificial Intelligence and the subsets

1.6 Machine Learning and its types

Machine Learning algorithms can be trained using three prominent methods: Supervised Learning, Unsupervised learning and Reinforcement Learning.

In Supervised Learning, we have prior knowledge of the output values of training samples. We have labelled inputs. Hence, the major goal in this type of learning is to learn a function, given a sample of data and desired outputs, that best approximates the relationship between the input and output variable [5]. Supervised learning can be further divided into Classification and Regression problems.

Classification problems are when the output variable is category which can be a binary like if you have ‘a disease’ or ‘not’ or a number of categories like predicting different colours ‘red’, ‘blue’, etc. Regression problems [6] are when the output data is a real value. Eg. housing prices, weights, etc.

In Unsupervised machine learning [7], the input dataset is unlabeled. Here the goal is to model the underlying structure or distribution of data so that the model can learn more from the data. Algorithms are left on their own to learn the interesting structures in the data. It is further divided into clustering and dimensionality problems.

In a clustering problem [8] the aim is to find the inherent groupings in the data like grouping the customers according to purchase behavior. In dimensionality problem [9], the model discovers rules for large portion of data like people who buy X also tend to buy Y.

In Reinforcement Learning, trial and error method is used to come up with a solution for the problem [10]. To enable the machine to do what we want, rewards or penalties are awarded according to the action it performs. The goal is to maximize the total reward. The model does not have any clue on how to solve the puzzle. It is upto the model on how to perform the task and maximize the reward starting from totally random trials to finishing with sophisticated tactics. The input is an initial state from which the model starts. There can be many possible outputs as there are variety of solutions to the problem. The model is trained on the input, it returns a state and the user decides whether to reward or punish the model based on the input. The model keeps learning and the best solution is decided based on maximum reward. Figure 1.3 shows the types of

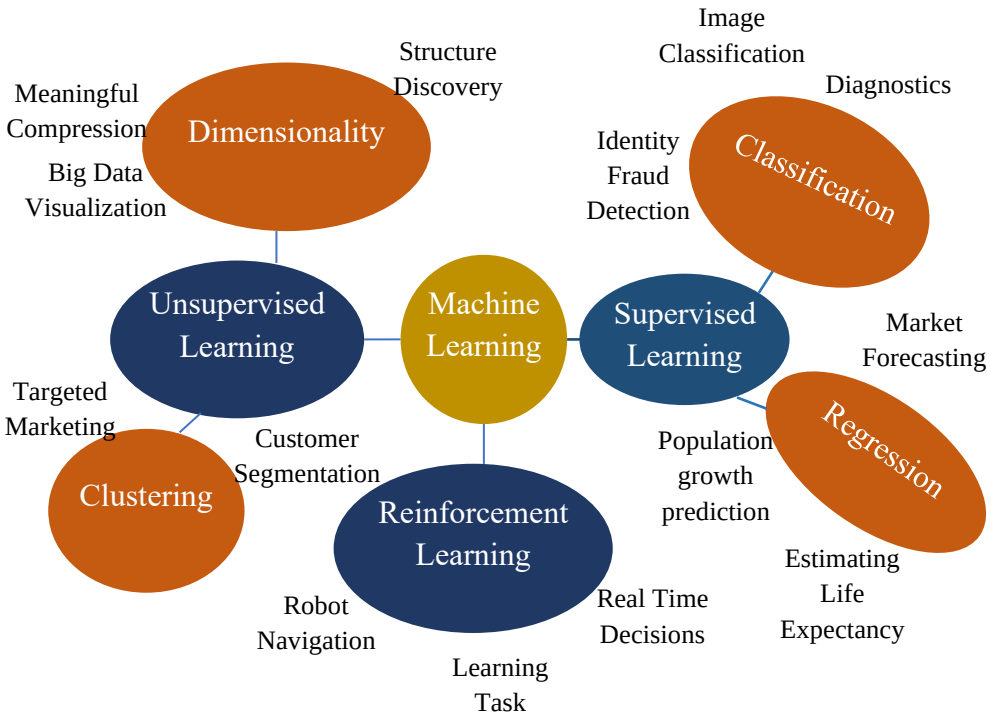

Figure 1. 3: Types of Machine learning

The problem we have in hand is a regression problem and uses supervised learning method where the power and performance values are predicted based on the input data from previous generations of processors.

1.7 Motivation

Estimating power and performance for the software stack running real-world applications is a key aspect to pre-Si system-level Hardware design. Current techniques are either fast enough to run the software stack but are quite inaccurate or provide accurate projections but are slow. With increasing accuracy, the speed of the simulators falls off. In the cycle-accurate (CA) performance simulators, the speed of the simulator typically drops to the extent where it is difficult to run actual software on top of the system [11] [12]. The new processors would be used on sophisticated applications which require the need to run long running workloads to estimate power and performance and yet provide feedback in shorter time. Thus, we can identify a gap in the present modelling methodology. We need a faster yet accurate model by which long running workloads can be run easily in shorter time and hence the power and performance can be estimated for the particular hardware configuration.

1.8 Objective

We propose a methodology here which makes use of Machine Learning techniques in order to predict power and performance of processors in a shorter time with fair accuracy. It extends the concept of using performance monitoring counters [13] [14] which can be broadly divided into hardware and workload features, to estimate power and

performance. We also have looked at an Automated ML solution which can be an extension to this work to have the whole system in place on a larger scale and we have explored the feasibility of using this solution. Thus, the proposed flow will result in early performance and power projection and reduces the turnaround time for the projection. It can also scrape out the need to develop performance simulators with each generation of processors which is in itself a time-consuming process. Instead here we untap the potential of previous generation data of the processors via Machine Learning Techniques to produce an effective methodology for projection of power and performance.

1.9 Organization of Thesis

The rest of the thesis is organized as follows:

Chapter 2 covers the review of past works on power and performance estimation and the proposed methodology. Chapter 3 explains how hardware and workload features can be used to predict performance of the processor using Machine Learning Techniques and how these features have a direct correlation in impacting the power and performance. Chapter 4 presents the Power estimation using the performance monitoring counters and how the Machine Learning model can be tuned to increase the prediction capabilities. Chapter 5 introduces the Automated ML modelling solution which offers a wide range of capabilities and the feasibility of using this solution to estimate power and performance. Chapter 6 covers the results achieved using this methodology to predict the power and performance as well as the results using the automated ML solution. We have also seen how the different categories of

workloads respond differently to the Machine Learning model. Chapter 7 explains the scope of future work in this area and the conclusion.

Chapter

2. Review of past work and problem formulation

Power and performance projection are important part of SOC lifecycle. Hence the techniques used for these are widely explored.

2.1 State of the art technologies for Performance Estimation

There are different methods existing to perform power and performance estimation. The tradeoff here is between accuracy and speed.

2.1.1 Performance Simulators

C++ based simulation environment or System C environment [15] are used for detailed processor performance stimulation using real life benchmarks. Many works have been developed in this space like building a System C framework for design space exploration and processor simulation [16]. Co-simulation methodologies using System C as hardware modelling language and on an Instruction Set Simulator as model of the processor has been explored [17]. The disadvantage with this approach is that it is very time consuming as large number of instructions are executed per benchmark. Each benchmark would contain a several millions of instructions. Thus, simulation of such large amount of instruction counts would take several hours or days even on today's fastest machines.

2.1.2 Analytical Models

They use algebraic sub-models and analytical formulas to estimate power and performance. These provide an insight into the interaction between the application and processor. They might use simple mathematical equations for a particular part of the design [18] to complex processor models with numerous formulas. They are much faster than cycle level simulation. There are two steps involved in estimation an application performance on a particular processor configuration. First step is to profile the application and the analytic model is applied on the top of the profiled data [19]. The next step is to apply mathematical formulas. Though this is a faster approach, accuracy is much lesser than cycle accurate simulators. If the model has to be more accurate, there is a need to formulate more complex formulas and hence it leads to a time-consuming development phase. Also, every design change poses the risk of complete invalidation of the postulated formula and hence it is much difficult to start all over again. We can see processor models directed at performance evaluation in [20] and [21]. However, in [22] the process of formulation of such a model and the enormous efforts behind it becomes evident.

2.2 State of the art technologies for Architectural power modelling

It is known that design-based power estimation is more accurate than architectural power modeling. But design-based power estimation is very compute-intensive and requires stable RTL, which is not available until late in the project. Hence architectural power modelling is used to deliver usable results

very early in the lifecycle of a project [23]. In order to best incorporate available information from design-based tools, architectural power models are typically designed in an evolutionary chain, with the power model for the newest product being represented as a set of deltas against the previous product's power model. At the oldest product in the chain, a power model built via regression against a trusted design-based power model is used. As products move forward in their lifecycle, gradual improvements to the quality of the power model are made and automatically carried through to subsequent generations in parallel development. The older power models are gradually changed into models which are directly regressed against a design-based power model. As a result of this methodology, for much of their lifetime a power model is actually defined in terms of scaling factors or features, which represent deltas against the previous power model. The best solution to estimate power using this methodology is to build power estimators into cycle accurate simulators which are used to understand the effect of architectural choices in performance [24].

2.3 Current Workflow

The most commonly used flow for architectural performance and power projection are performance simulators [25]. The input for the simulator are the workloads which are large instruction traces [26] and the output from simulators are the stat files which are fed into a database.

A configuration is called an experiment. A study contains one or more experiments. Studies that run on performance simulator produce stat files. An experiment contains all the simulator results for the traces ran on the corresponding configuration (jobs). Stat files include stats of different

metrics, for many traces, and many experiments. Figure 2.1 describes the study structure.

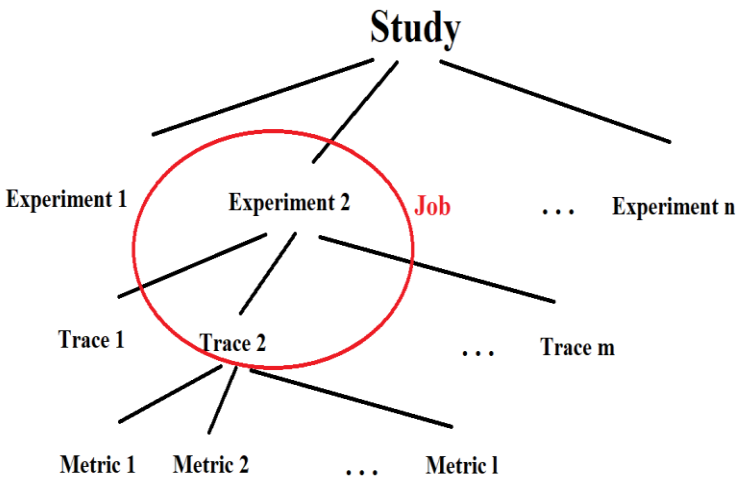

Figure 2. 1: Study Structure

The simulator outputs stat files for each job, which include many metrics, as well as logfiles describing the status of the job run on the simulator. Stat files and log files are the input of the database management solution. Simulators usually run a trace list (N traces) on various configurations (M cfgs). Each run is called a job (ie.1 trace on 1 cfg, total: NxM jobs). Each job run produces a stat file with various metrics. These stats are uploaded into the database management solution. Figure 2.2 is a simplified diagram of how power and performance is projected using the current flow.

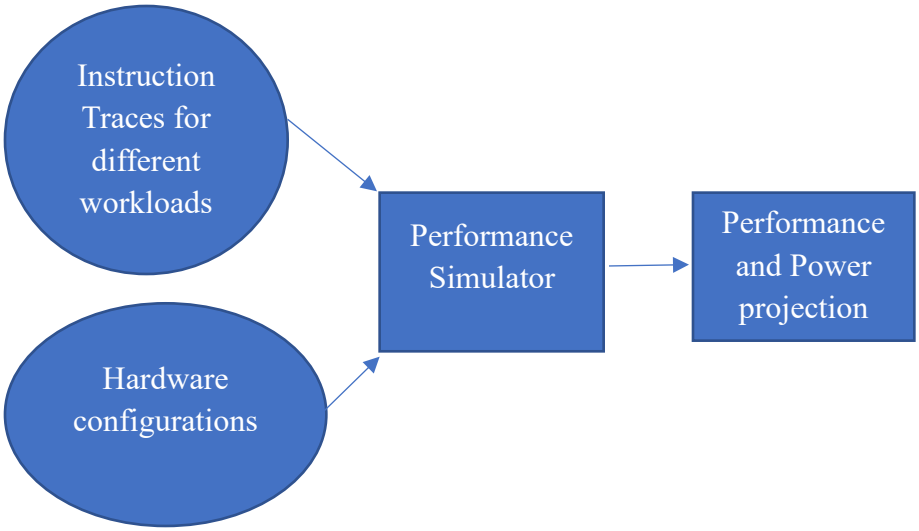

Figure 2. 2 Current Workflow

2.4 Proposed model

A particular set of configurations and workloads would produce a given performance metric and power. The most commonly used metric for performance measurement is the instructions retired per cycle. In this method the input data is a combination of workloads and several metrics which has been obtained from the previous generations and we apply ML algorithms over this data to estimate power and performance. The target would be any of the performance metric like instructions retired per cycle or power. The inputs or features affecting the target will be workloads and configurations. Thus, the target can be predicted for a given set of workload and configurations. The result is compared to enterprise solution which automatically provides a leaderboard of ML models. Finally, we will see how this methodology performs at par with the state-of-the-art technologies.

Chapter

3. Performance Estimation using Machine Learning Techniques

3.1 Dataset

Design architects try different configurations on benchmarks or workloads and project power and performance at an early stage of RTL to make chips that would meet the product requirements. Studies run daily on performance simulators thereby producing thousands of stats files. Thus large magnitude of data is produced. This data is stored in a system used for architecture performance work making the storage, management, analysis and visualization of such large amounts of data easier. The data used for this model is taken from this system. Figure 3.1 shows the how the data is handled in the machine learning process.

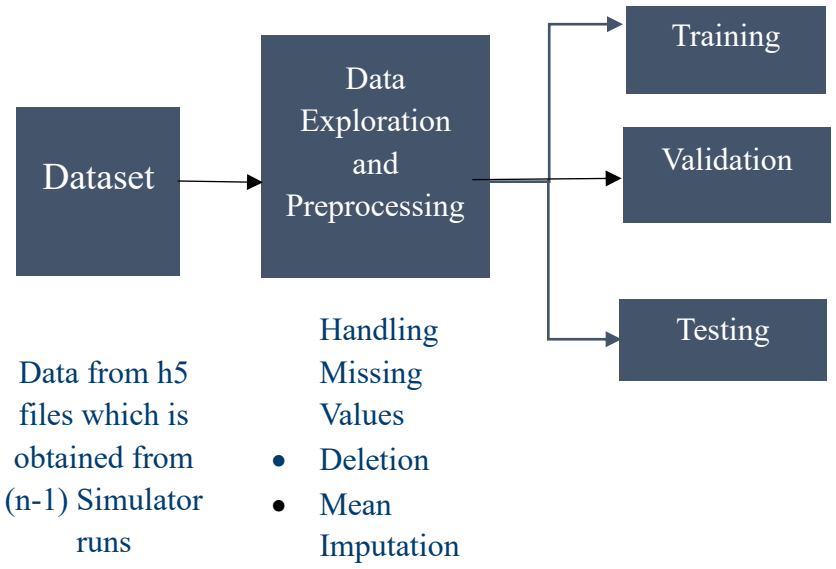

Figure 3. 1: Machine learning steps

3.2 Training, Validation and Testing Set

There were 1436 traces in total. Out of which 677 traces were used for training and for validation. Using this, the model was trained and used to project the data for testing set which consist of 759 traces. These traces were also split according to the eleven different categories in order to see the sensitivity of each category to prediction. Figure 3.2 shows the classification of data into three different categories during the machine learning process.

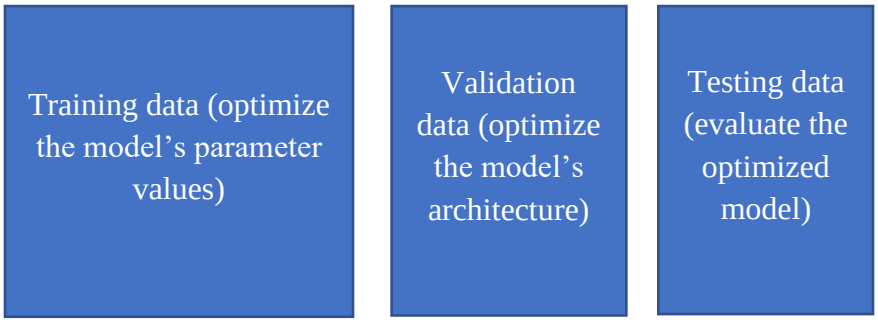

Figure 3. 2: Classification of data in Machine Learning

3.3 Target

The metric chosen to measure performance here is IPC (Instructions retired per cycle). It is one of the major aspects of a processor's performance. It is the average number of instructions executed per clock.

Performance $\propto$ 1/(run time)

1/(run time) = (Frequency*IPC)/instruction count

An increase in IPC indicates an improvement in processor performance. It usually ranges between 0.3 to 4. It is an excellent metric to judge the overall performance of a

processor with respect to different applications. An IPC of 1 is considered acceptable for HPC applications but the expected IPC varies according to the different application domains. Its value is affected by various factors like memory stalls, instruction starvation, branch miss prediction, cache misses and long latency instructions.

3.4 Metrics

The features that would help in target prediction are chosen from a set consisting of large number of metrics. The total set of metrics was divided into into:

1) Workload dependent features:

   These features do not change from generation to generation. Their values are dependent on the particular benchmark only. It is independent of the change in processor configurations. Examples for these features are int_32b, execution_count_simd_fp, etc.

2) Hardware dependent features:

   These features changes from generation to generation depending on the processor hardware configurations. Example: icache misses, l2_hit, dcu_miss,etc.

3.5 Features

The features chosen to predict the IPC are as follows:

3.5.1 Hardware dependent features

3.5.1.1 Baclear:

It estimates the fraction of cycles lost when an early branch prediction is corrected by a later branch prediction. The branch prediction unit (BPU) is unable to provide correct prediction and it is corrected by other branch handling mechanisms. It happens when the code has many branches that can't be consumed by BPU.

3.5.1.2 alloc_window:

This metric measures how memory is allocated for the window of a given size. It is given by allocate / cycles_uop_allocated.

3.5.1.3 dtlb_hit:

It estimates the number of times the first-level data TLB (DTLB) is hit. The memory contains a page table which maps virtual and physical memory. To reduce reference to these recently used portions of these are cached in Translation-look-aside buffers (TLB) which are consulted for every virtual address translation. Similar to data caches farther the request hast to go to get satisfied worse the impact on performance.

3.5.1.4 icache_miss

Instruction cache misses are failed attempts at reading data in cache which results in main memory access leading to longer latency. They cause the largest delay because the processor has to wait till instruction is fetched from main memory.

3.5.1.5 l2_hit

It is the last and longest latency level in memory hierarchy before the main memory is accessed. They incur a performance penalty.

3.5.1.6 stlb_hit

Loads that hit the second level data translation buffer.

3.5.1.7 all_branch_retired

Number of branch instructions retired for all branch types.

3.5.1.8 itlb_miss

It is the number of page walk requests due to instruction translation look aside buffer misses.

3.5.2 Workload dependent features:

3.5.2.1 skl_execution_count_simd_fp

It is the count of single instruction, multiple data floationg point instructions.

3.5.2.2 int_32b

It is the share of int 32b uops executed. It is given by skl_execution_count_int_stack_32b / execution_count.

3.5.2.3 int_64b

It is the share of int 64b uops executed. It is given by skl_execution_count_int_stack_64b / execution_count.

3.5.2.4 x87

It is the share of x87 uops executed. It is given by skl_execution_count_X87 / execution_count.

3.5.2.5 vec

It is the share of x87 uops executed. It is given by skl_execution_count_simd_fp / execution_count.

3.5.2.6 INST_FDIV

There is an integer divide functional element within the integer unit. This is the count of instruction required in executing a floating point division.

3.6 Flow

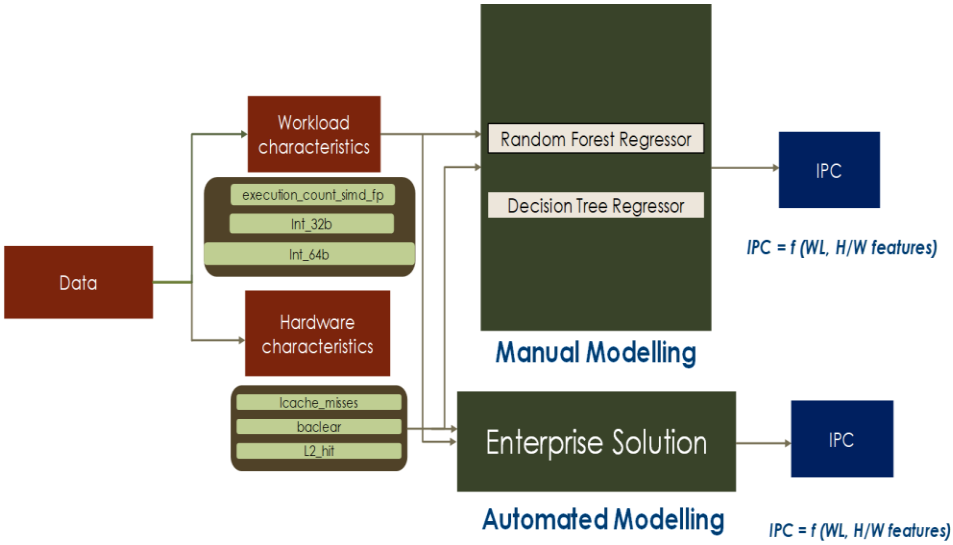

Figure 3. 3: Proposed Workflow for Performance Estimation

Figure 3.3 describes the whole flow used in this system. The hardware and workload features associated with the different workloads are fed to the machine learning models as well as an automated ML solution. This data is used to train the model. Using this model, we can predict the performance of the next generation processors.

3.7 Machine Learning Steps

1. Problem formulation
2. Data Collection
3. Data preparation
4. Model Selection
5. Training the model
6. Evaluation of the model
7. Hyperparameter tuning
8. Making predictions

3.8 Data Exploration and Preprocessing

The main steps involved in data preparation are data formatting, cleaning and sampling [27].

3.8.1.1 Data Formatting

If the data is not available in a format you can work with, this step is used. The data was present in the relational databases was converted into an excel format so that machine learning steps could be done easily on it.

3.8.1.2 Cleaning the data

Not all data instances carried full data. Hence this missing data has to be fixed first. There are several methods that can be used here like deleting the rows which contains missing data, mean/median imputation and predicting the missing values. Deletion of rows is the most most simple approach, but the disadvantage is that it reduces the sample size. We have used mean imputation here where the mean value of the particular metric which had empty rows was calculated and the mean was imputed for the missing rows.

3.8.1.3 Sampling the data

Sometimes there would be far more data than necessary, and we need to take only the necessary data otherwise it would lead to longer running times reducing the efficiency of the machine learning model.

3.9 Model Building

In machine learning, it is important to choose a correct model for training. A model is a mathematical representation of a real-world process. This model is provided with the training dataset and the model learns from it. Then validation is done on validation dataset from the results of which the best model is chosen from. Here the model used is Random Forest Regressor. Also, the training dataset was fed into an automated enterprise solution for machine learning which give us a dashboard of the best models that can be used for the particular problem depending on the validation score.

3.10 Decision Tree Regression

It builds the regression models in the form of a tree structure [28]. It breaks down the dataset in to smaller and smaller subset and the decision tree is incrementally developed. It is arrived at a result by asking a series of questions to the tree, each question reducing the set of values that can be a possible output until the model gets confident to make a single prediction. The model determines the content and order of questions. The result is a tree with decision nodes and leaf nodes. A decision node has two or more branches. Leaf node represents a decision. The topmost decision node

in a tree is called root node. It forms an excellent foundation for various other models like random Forest [29]. Figure 3.4 show the Decision Tree Regression Model.

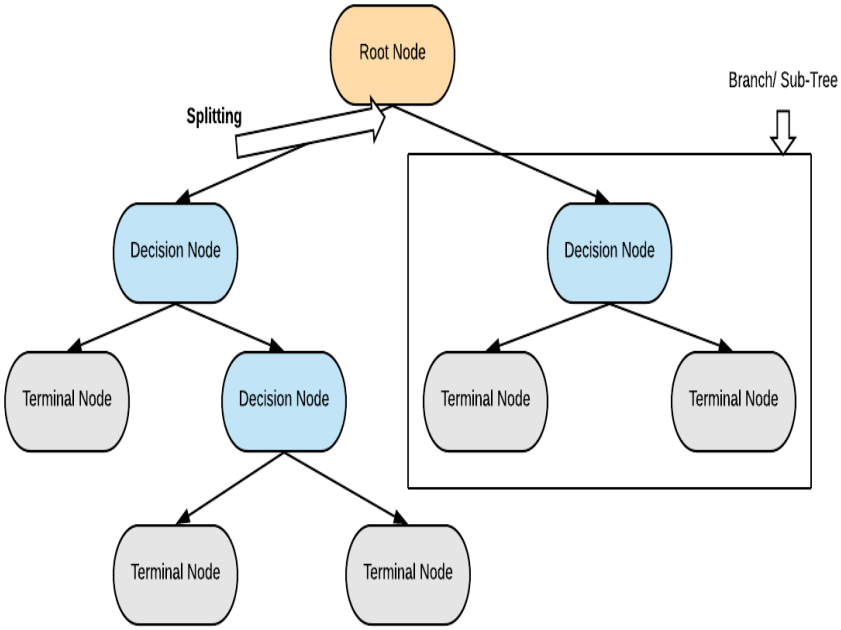

Figure 3. 4: Decision Tree Regression

3.11 Random Forest Regression:

It is a supervised machine learning algorithm which depends on ensemble learning. Ensemble learning means different algorithms are joined or you repeat the same algorithm several times to build a more powerful prediction model. The Random Forest algorithm combines several algorithms of same type, i.e. Multiple decision trees and hence a forest is created and hence the name Random Forest [30] [31]. The basic steps involved I Random Forest algorithm are:

1. Pick N random data instances from the dataset.

2. Build a decision tree based on these instances.
3. Choose the number of trees you want and repeat steps 1 and 2.
4. For Regression problem, each tree in the forest predicts an output Y for each record. The final value is decided by taking average of values predicted by all the trees in the forest.

3.12 Evaluation of the model

Evaluation of the model was done using RMSE (Root Mean Square Error). Root mean square is a standard measure to calculate the error in predicting the target in a machine learning model. The lower the RMSE the better your model has learnt [32].

$$RMSE = \sqrt{\frac{\sum_{i=1}^{N}(Predicted_i - Actual_i)^2}{N}}$$

It measures the standard deviation of predicted values against actual values.

3.13 K- Fold Cross Validation

Performance of a model increases as the size of training set increases. Also, the model performance estimates are more consistent when the validation set is large. Hence it is advisable to use as much data possible in validation and training. Hence, we use the cross-validation method to maximize the data available for each of these sets [33]. This process involves:

1. Splitting the data into two or more sections called folds.

2. Creating one model per fold, the data assigned to that fold is used for validation and the rest is used for training.

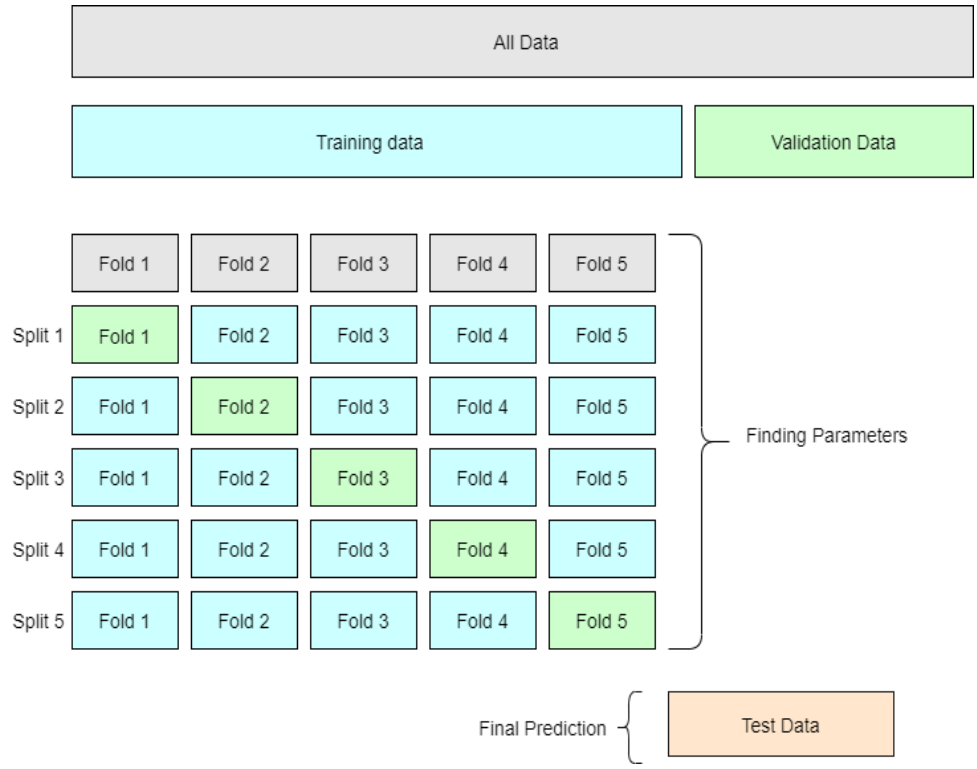

Figure 3. 5: K- Fold Cross Validation

Advantage:

It gives better estimate of the model performance.

Disadvantage:

It has multiple passes and hence it is computationally expensive and takes longer to run.

3.14 Making predictions

The predictions were made on test data which was a complete Blackbox to the machine learning model. The test data consisted of 759 data instances whose IPC were predicted.

Chapter

4. Power Estimation using Machine Learning Techniques

4.1 Overview

Power is computed by multiplying a set of event costs (EC) by the corresponding performance simulator derived activity factor (AF). ECs are organized in a hierarchy with the FUBs forming the lowest level and proceeding upwards through Units, Clusters and finally Core as a whole.

$$Fub's\ power = idle + \sum_{i=1}^{n} AF_i * EC_i$$

Power is usually computed as dynamic capacitance, C_{dyn}, which is independent of operating voltage and frequency. AFs are unitless ratios in the range [0,1] that represents the portion of cycle in which a particular activity occurs. ECs are the sum of dynamic capacitances associated with a single occurrence of a particular activity. Each fub is also assigned an idle EC, which is the dynamic capacitance produced every cycle corresponding to an AF of 1. The idle EC represents the free running clocks.

4.2 Dataset

The data is obtained by running the same set of benchmarks used for power estimation on a tool which does Architectural

power modelling for the core. Thus, the power numbers corresponding to each workload was obtained.

4.3 Training, Validation and Testing Set

The traces used for power estimation are the same as that of performance estimation, thus 1436 traces were used. 677 traces were used for training and for validation. This data was used to train the model and the target was predicted for 759 traces which forms the testing set.

4.4 Target

The target was the power numbers for each particular workload.

4.5 Features

The features used to predict the power were the hardware dependent and workload dependent metrics. Some metrics which were used to predict the performance were reused here and some new metrics were added.

4.5.1 Hardware dependent features

4.5.1.1 dtlb_hit

4.5.1.2 all_branch_retired

4.5.1.3 alloc_window

4.5.1.4 baclear

4.5.1.5 l2_hit

4.5.1.6 stlb_hit

4.5.1.7 itlb_miss

4.5.1.8 cycles

It indicates the number of cycles required to complete a particular instruction.

4.5.1.9 Instructions retired:

It is an important hardware performance event and calculates how many instructions were completely executed.

4.5.1.10 mlc_data_read_for_ifu

IFU stands for instruction Fetch Unit. It fetches up to 16 bytes of instruction bytes each cycle from the instruction cache to the instruction length decoder (ILD).

4.5.1.11 execution_count_wb

It is the execution count for writeback caching (WB). Any new processor data is written to the cache and not in the memory. The memory write process is only performed when the cache data needs to be edited or purged for new content.

4.5.2 Workload dependent features:

4.5.2.1 skl_execution_count_simd_fp

4.5.2.2 int_64b

4.5.2.3 vec

4.5.2.4 int_32b

4.5.2.5 x87

4.6 Flow

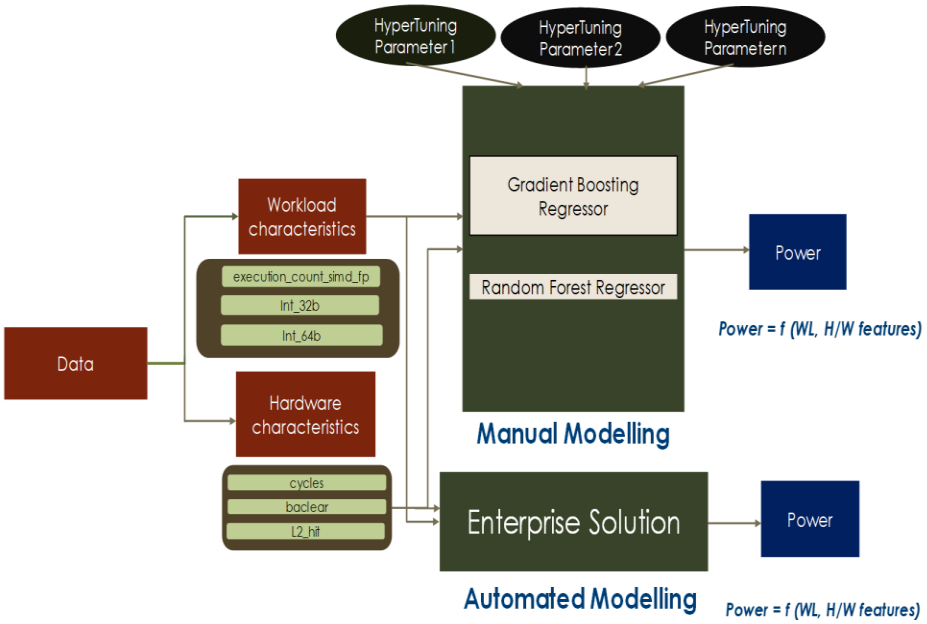

Figure 4. 1: Proposed Workflow for Power Estimation

Figure 4.6 illustrated the flow used to estimate power using machine learning techniques. It uses the hardware and workload features from previous generation processors to train the Machine Learning model and predicts the power of next generation processors for the different workloads which mimics the real-world applications.

4.7 Data Exploration and Preprocessing:

The major change here was in the data exploration an data preprocessing step to get the data in the format we required. The power data had to be combined with the data from performance simulator to map the power numbers to the

architectural and workload features. Once this was achieved the data was cleaned by imputing the null values by the mean of the particular feature.

4.8 Model building

Two models were used here for comparison: Random Forest Regressor and Gradient Boosting algorithm.

4.9 Gradient Boosting Regression

In boosting, each tree is fit on the modified version of original tree [34]. The baseline for this model is the AdaBoost algorithm [35] which puts more weight on those instances which are difficult to classify and less on those that are handled well. It begins by training a decision tree where all observations are assigned equal weight. After the evaluation of the first tree, weights are increased on those observations that are difficult to classify and less on those that are well handled. The second tree is grown based on this weighted data. The idea is to improve the predictions of the first tree. The classification error is calculated from this 2-tree ensemble model and we grow the third tree to predict the revised residuals. The same process is repeated for a certain number of iterations. Prediction s of final ensemble model is based on the weighted sum of output of previous trees.

The major difference in Ada boost algorithm and Gradient Boosting is the way in which they identify the shortcomings of weak learners. Ada boost identifies it by using high weight data points while gradient boost performs by using gradients in loss function. Loss function is a measure of how

good the model's coefficients are at fitting the underlying data.

4.10 Hyperparameter Tuning:

Hyperparameters define the model architecture. These cannot be learnt from the training process. They are fixed before training begins. They explain the complexity of the model and their parameters like how fast it should learn [36].

4.10.1 Hyperparameters used for tuning Gradient Boosting Regressor:

n_estimators: It indicates the number of boosting stages to perform. Since gradient boosting is fairly robust to overfitting, hence large number leads to better performance. Default value is 100.

learning_rate: It determines the impact of each tree on the final outcome. Gradient Boost starts with an initial estimate, this keeps on updating according to the output of each tree. The learning rate controls the magnitude of this change in the estimates. Default value is 0.1.

max_depth: It limits the number of nodes in the tree. Default value is 3.

Subsample: The fraction of samples to be used for fitting the individual base learners. Default value is 1.

4.10.2 GridSearchCV

We have used the approach of GridSearchCV for hyperparameter tuning. It looks through each combination of hyperparameters. It generates candidates from a grid of parameter values specified. A score function is taken to evaluate the parameter setting. We have chosen mean squared error here.

4.10.3 Optimized parameters

Hyperparameter	Default Value	Optimal Value
n_estimators	100	2000
subsample	1	0.5
max_depth	3	4
learning rate	0.1	0.01

Table 4. 1: Optimized hyperparameters of Gradient Boosting Regressor

Table 4.1 shows the hyperparameters obtained after tuning the model. The model was again trained with the new optimized parameters and results were calculated on the basis of this new model.

Chapter

5. Power and Performance Estimation using automated ML Modelling

5.1 Introduction

It is a predictive analytics automation platform to rapidly build and deploy predictive models. It streamlines the data science process, leading to faster results and fewer integration steps. Figure 5.1 shows the steps involved in the Automated ML modelling process.

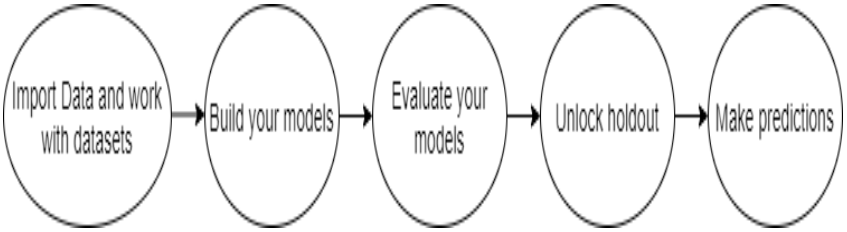

Figure 5. 1: Steps involved in Automated ML modelling

5.2 Advantages

1. Data Exploration/ Analysis
2. Model recommendation
3. Time to value
4. Prediction accuracy
5. Data ingestion
6. Data Export
7. Model deployment
8. Model export

5.3 How it works?

To build an accurate predictive model it is required to search through a nearly infinite combination of data transformations, models, features, algorithms and tuning parameters. This Enterprise solution simplifies model development by performing a parallel heuristic search for the best model or ensemble of models, based on the characteristics of the data and the prediction target. By cost-effectively evaluating thousands of models in parallel across a large cluster of servers, the solution delivers the best predictive model in the shortest amount of time.

The common predictive modeling workflow is to perform an exploratory data analysis (EDA), select a target feature to predict, select a performance metric, and search for the algorithm to model the domain. With the solution, the algorithm search is automatically performed for you.

- Import data and work with datasets
- Optionally, set advanced options
- Build your models
- Evaluate your models
- Unlock holdout
- Make predictions

5.4 Importing the data

Importing the data can be done from a local file, an external datasource, from a URL or from HDFS.

5.5 Exploratory Data Analysis and Model building

In this stage we select the target which needs to be predicted and the solution starts analyzing the data we provided and creates summary statistics based on this data. The feature correlation with the target is also performed in this phase. After this model building begins and we can choose an optimization metric of our choice (like RMSE) to evaluate the performance of the model. Once this phase ends the solution comes up with leaderboard which showcases the best models for our problem statement.

5.6 Leaderboard

This is one of the most important feature in this solution where it exhaustively search in its repository for the best model that would suit the data we provided and then builds up a leaderboard of different models depending on their ranking with respect to any validation metric like RMSE, Gini Norm, MAE, R squared, etc. New models blending already existing models are also used in this process. It gives us suggestions on which model is best for deployment, which is the most accurate model, and which is the fastest model. Figure 5.2 shows an example of how leaderboard looks like.

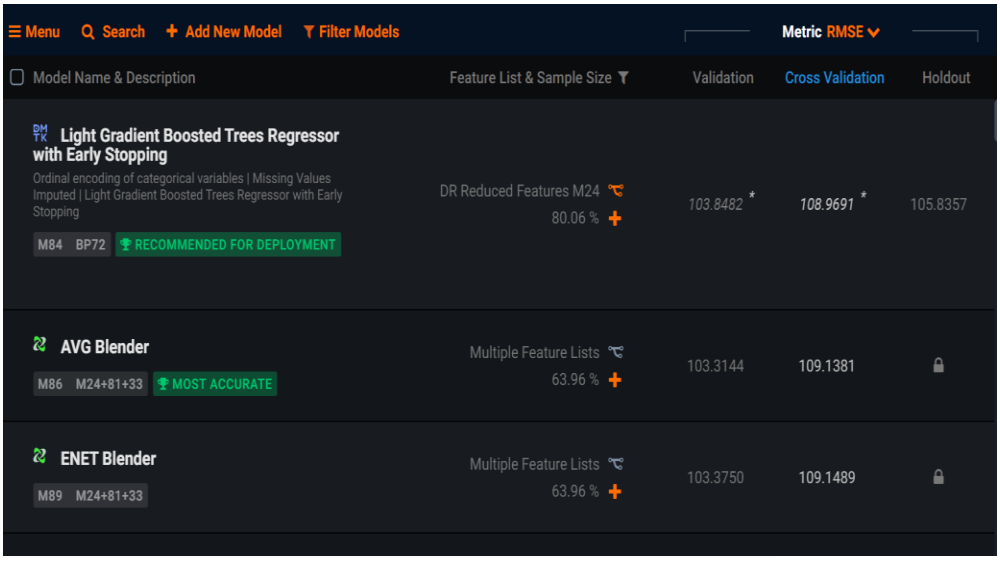

Figure 5. 2 Leaderboard of models

5.7 Disadvantages

The problem with this automated solution is that we have very less control in the overall machine learning process. It is difficult to choose the internal model characteristics like hyperparameters of our choice. Another main problem with this approach is that it provides less debuggability. It is difficult to know the exact reason why a particular model came at the top of the leaderboard i.e. what characteristics or pattern in the data made it at the top or bottom of the leaderboard.

Chapter

6. Results and Discussion

In this chapter we have discussed the results of the proposed method. The RMSE of various methods, error graph and error area are plotted.

6.1 Performance Estimation

6.1.1 RMSE Values

Decision Tree Regressor	Random Forest Regressor	Automated Solution
1.033	0.88	0.84

Table 6. 1: RMSE Values of different models for performance estimation

6.1.2 Error graph

The grey lines in the graph indicates the deviation of predicted values from actual target values. The x axis shows the different workloads and y axis shows the predicted IPC values.

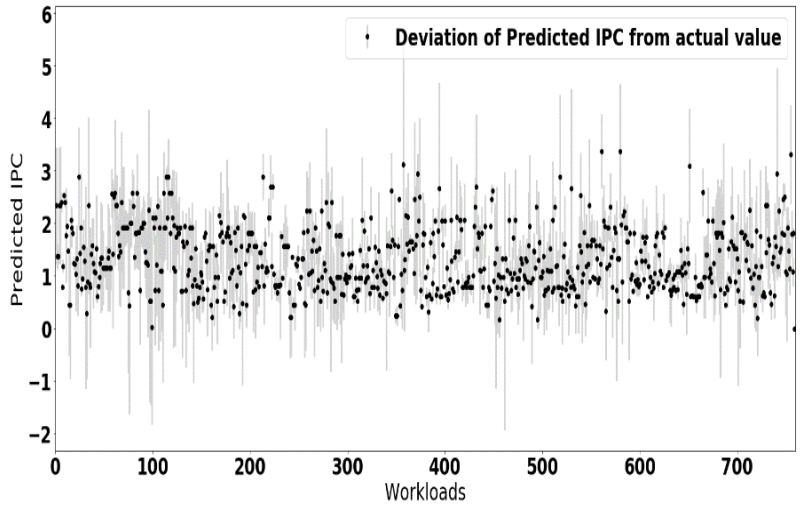

Figure 6. 1: Error Graph of Decision Tree Regressor

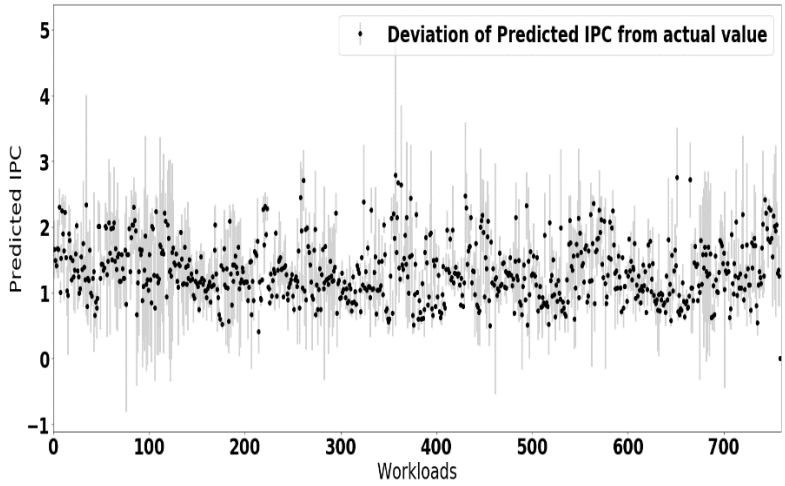

Figure 6. 2: Error graph of RandomForest Regressor

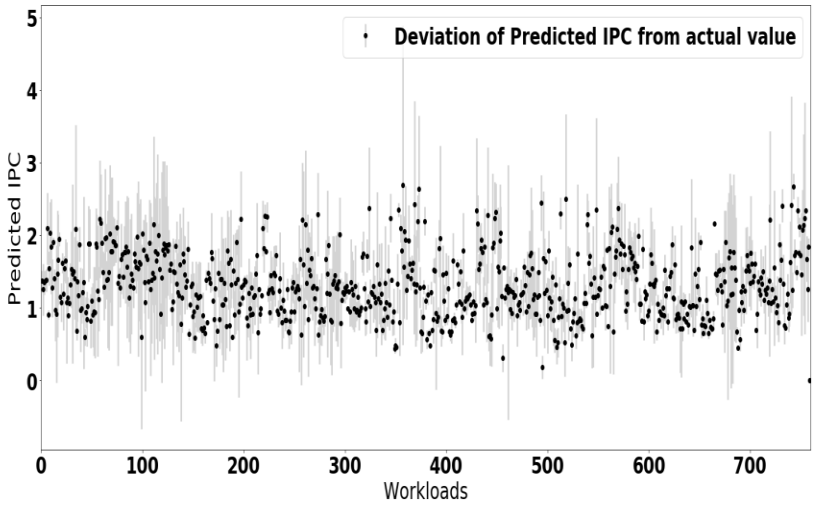

Figure 6. 3: Error Graph using Automated ML solution

6.1.3 Error Area graph

This graph indicates the deviation of predicted values from actual values in terms of area. The larger the error area less accurate is the prediction. Here x axis denotes a sample of workloads axis denote the predicted IPC values.

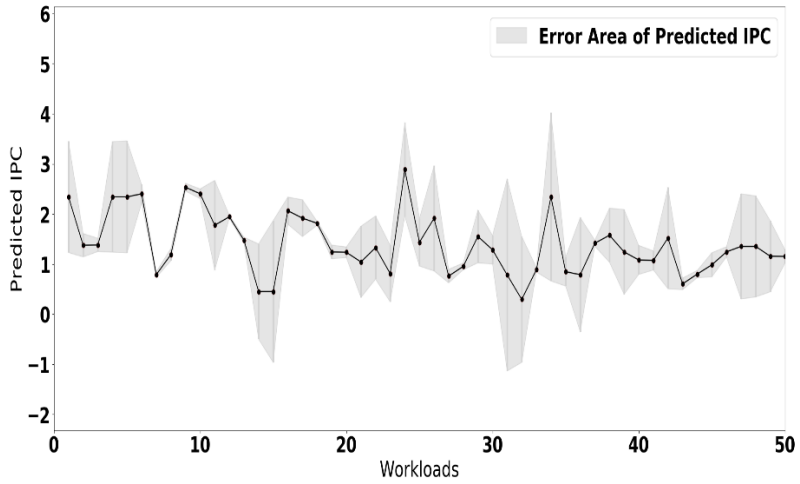

Figure 6. 4 Error Area Graph of DecisionTreeRegressor

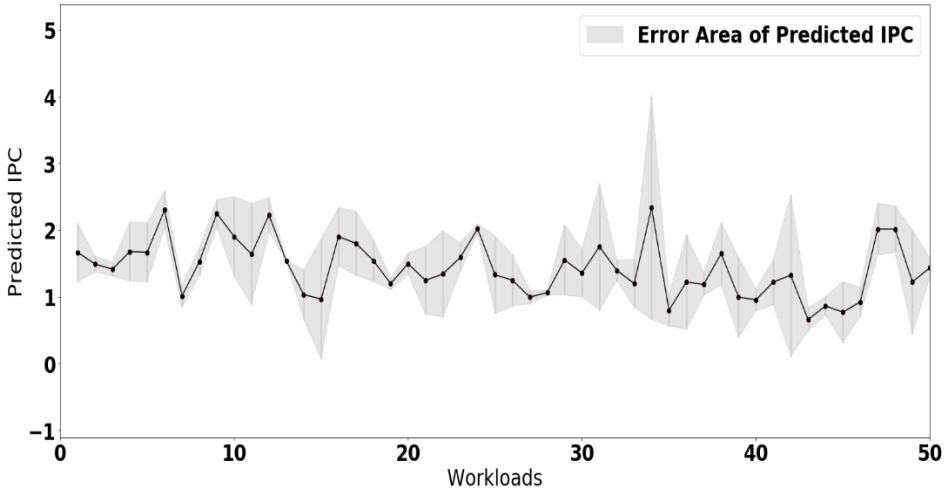

Figure 6. 5: Error Area Graph for Random Forest Regressor

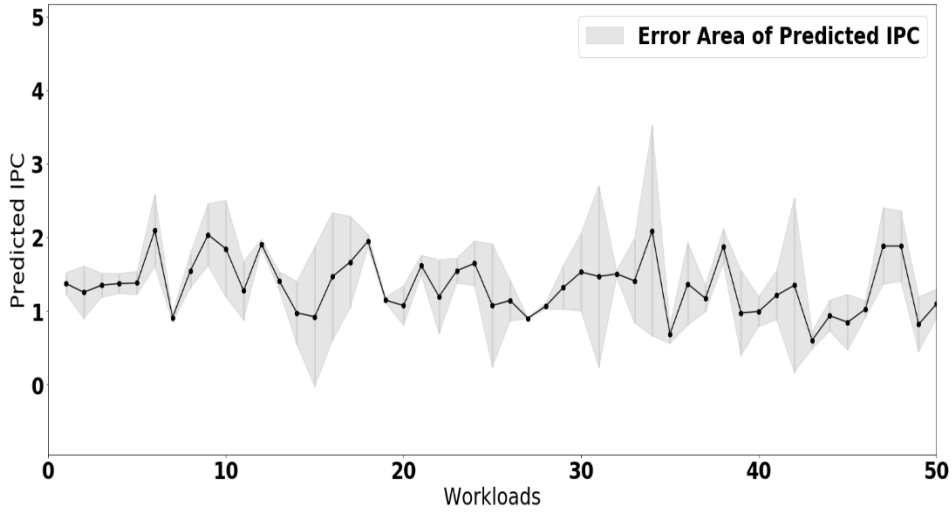

Figure 6. 6: Error Area Graph using Automated ML solution

Thus, we can see that Random Forest Regressor performs well here among our models.

6.1.4 Sensitivity of different categories of traces to the model

The sensitivity of different categories of traces to the model (Random Forest Regressor), i.e. how prediction accuracy varies depending on the category is calculated. RMSE of the predicted values of each particular category of traces is used as a measure to know how well the categories respond to the model.

CATEGORY	RMSE
AppleSAW	1.41714
FSPEC17	0.613354
ISPEC06	1.040431
ISPEC17	0.748692
SYSmark	0.92
client	0.990454
embedded	0.905956
games	0.727266
kernel	2.29
multimedia	1.009672
FSPEC06 (whole category hidden from training)	0.929292

Table 6.2: Sensitivity of different categories to Random Forest Regressor for Performance Estimations

Here we can see that the category ‘kernel’ had the highest RMSE which means predicting IPC values for the category ‘kernel’ produced much lesser accurate results. This is because this category consisted of lot of outliers in the data.

6.2 Power Estimation:

6.2.1 RMSE Values

Random Forest Regressor	Gradient Boosting Regressor (After Hypertuning)	Automated Solution
137.01	114.4	109

Table 6. 3: RMSE Values of different models for power estimation

6.2.2 Error graph

The grey lines in the graph indicates the deviation of predicted values from actual Power values. The x axis shows the different workloads and y axis shows the predicted IPC values.

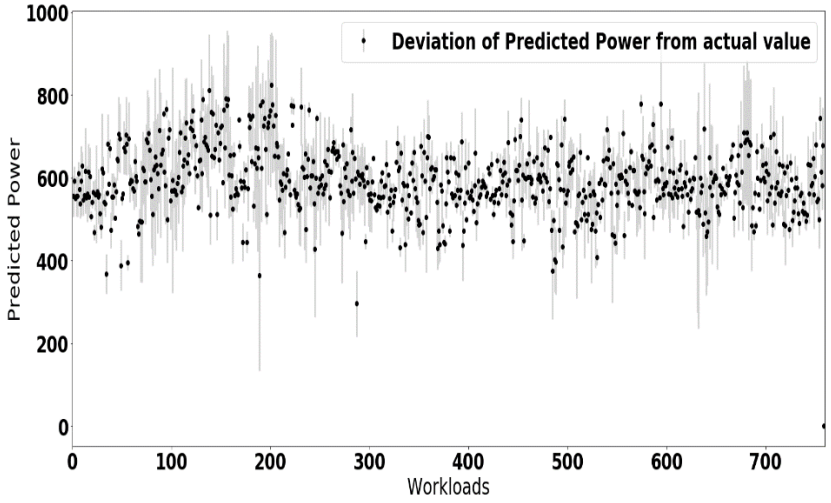

Figure 6. 7: Error Graph of Random Forest Regressor

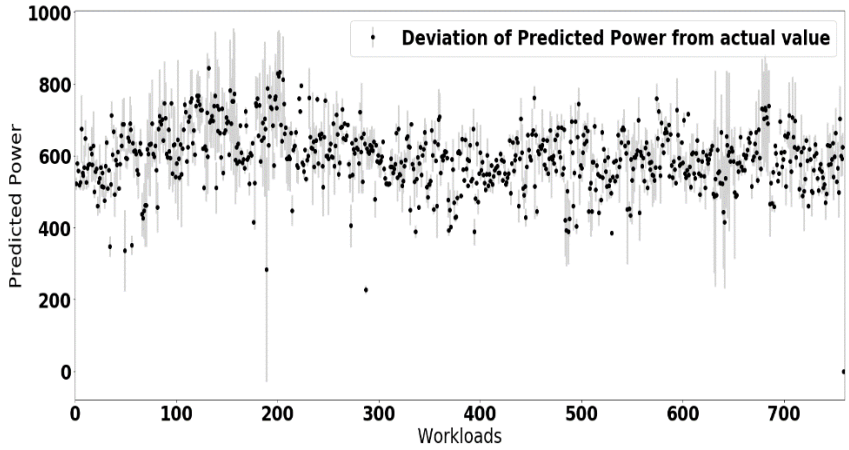

Figure 6. 8: Error graph of Gradient Boosting Regressor

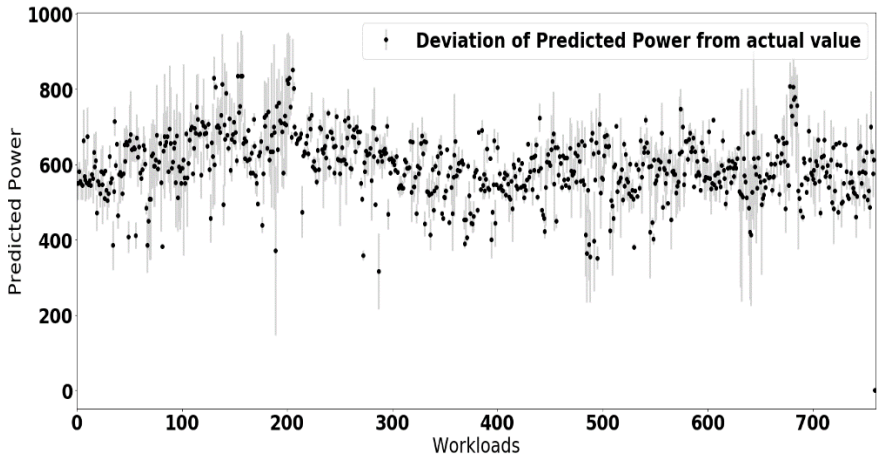

Figure 6. 9: Error graph using Automated ML Solution

6.2.3 Error Area graph

This graph indicates the deviation of predicted values from actual values in terms of area. The larger the error area less accurate is the prediction. Here x axis denote a sample of workloads and y axis denote the predicted Power values.

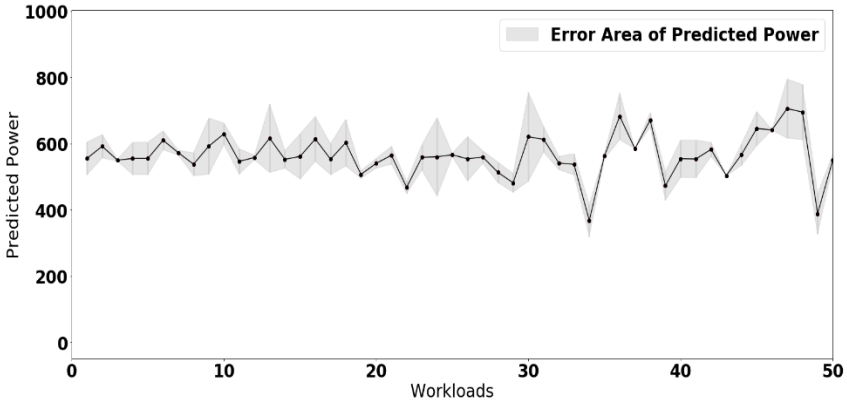

Figure 6. 10: Error Area Graph of RandomForestRegressor

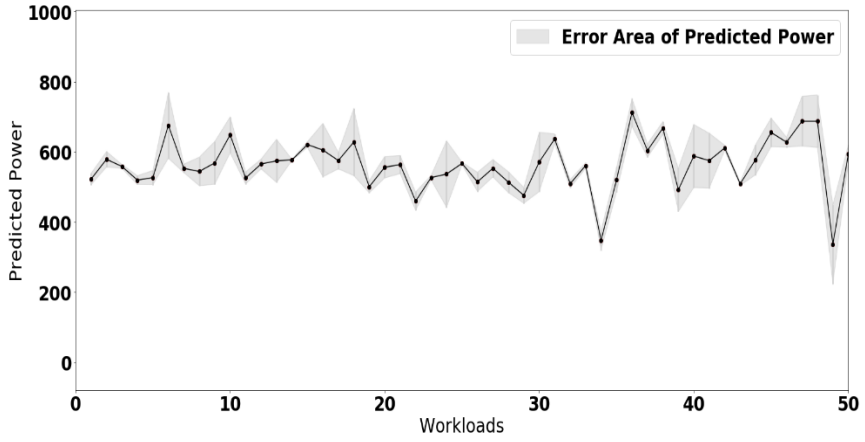

Figure 6. 11: Error Area Graph for Gradient Boosting Regressor

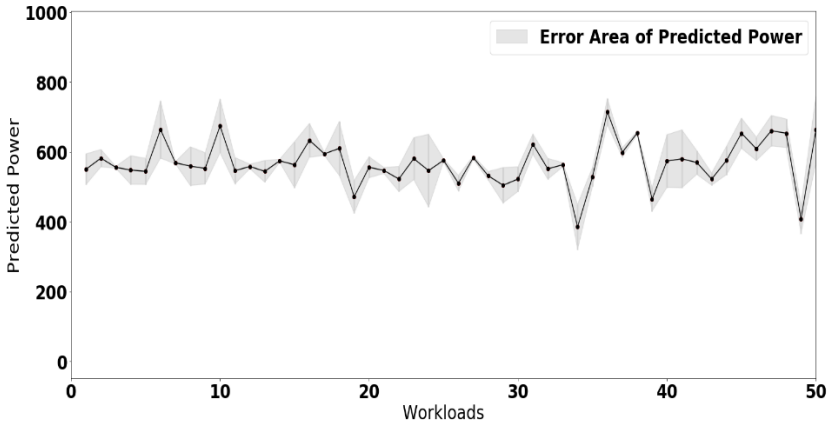

Figure 6. 12: Error Area Graph using Automated ML solution

Thus, we can see that Gradient Boosting Regressor performs better here among our models.

6.2.4 Sensitivity of the model to different categories of traces

This is the sensitivity of the model (Gradient Boosting Regressor) on how it predicts the target values according to

different categories of traces. RMSE of the predicted values of each particular category of traces is calculated.

CATEGORY	RMSE
AppleSAW	126.08
FSPEC17	94.41
ISPEC06	109.38
ISPEC17	74.94
SYSmark	67.07
client	96.2
embedded	92.54
games	151.7
kernel	262.46
multimedia	97.93
FSPEC06 (whole category hidden from training)	187.02

Table 6. 4: Sensitivity of different metrics to Gradient Boosting Regressor for Power Estimation

Here we can see that the category ‘kernel’ had the highest RMSE which means predicting Power values for the category ‘kernel’ produced much lesser accurate results. This is because this category consisted of lot of outliers in the data.

Chapter

7. Conclusions and Scope for Future Work

7.1 Conclusion

Power and Performance estimation at an early stage in SoC lifecycle is always the need of the hour. Here we have explored methods to estimate power and performance in a faster way and with fair accuracy making use of Machine learning techniques. This method can help to save lot of time because the traditional performance simulators and power estimation tools take a couple of hours to project the power and performance. Meanwhile here we utilize the previous generation data to project the power and performance for next generation.

7.2 Future Work

This machine learning technology can be further expanded by plugging the previous generation data to an automated ML solution which can provide a leaderboard of all suitable models from its repository, hence making it easier for us to choose which model can be used to predict the target. This can be made possible by developing a Python API which can automatically feed the data produced in any particular generation to the automated system from which the power and performance of the next generation processors can be calculated. There can be also a mechanism to identify the outliers in the data which potentially can reduce the accuracy

of the prediction and eliminate them to get better projection of power and performance data.

REFERENCES

[1] T. S. J. Abhijit Ray, "Practical Techniques for Performance Estimation of Processors," in *Proceedings of the 9th International Database Engineering & Application Symposium (IDEAS'05)*, 2005.

[2] G. Z. a. Z. K. H. Jie, "Research on evaluation method of electronic product maturity," in *IEEE 2nd International Conference on Computing, Control and Industrial Engineering*, Wuhan, 2011.

[3] V. Datla, "Software Performance Workload Modelling," *International Journal of Computer Applications Technology and Research,* vol. Volume 6, no. Issue 1, pp. 13-18, 2017.

[4] H.-D. Wehle, "Machine Learning, Deep Learning, and AI: What's the Difference?," in *Data Scientist Innovation Day*, 2017.

[5] N. T. a. A. S. A. Singh, "A review of supervised machine learning algorithms," in *3rd International Conference on Computing for Sustainable Global Development (INDIACom)*, New Delhi, 2016.

[6] J. Li, "Regression and Classification in Supervised Learning," in *International Conference on Computing and Big Data*, Taiwan, 2019.

[7] R. S. a. A. Abraham, "Comparison of Supervised and Unsupervised Algorithms for Pattern Classification," *International Journal of Advanced Research in Artificial Intelligence,* vol. Vol. 2, 2013.

[8] D. &. C. P. &. M. R. Greene, "Unsupervised Learning and Clustering," in *Applied and Computational Mechanics*, 2008.

[9] A. K. Cherukuri, "Analysis of unsupervised dimensionality reduction techniques," *Computer Science and Information Systems ,* 2009.

[10] W. Q. a. Z. Zhongli, "Reinforcement learning model, algorithms and its application," in *International Conference on Mechatronic Science, Electric Engineering and Computer (MEC)*, Jilin, 2011.

[11] R. S. V. P. R. a. B. J. Shang Li, "Rethinking Cycle Accurate DRAM Simulation," in *Proceedings of the International Symposium on Memory Systems*, New York, 2019.

[12] K. L. B. S. a. M. L. H. Cain, "Precise and accurate processor simulation," in *Workshop on Computer Architecture Evaluation Using Commercial Workloads*, Feb. 2002.

[13] X. W. a. V. Taylor, "Utilizing Hardware Performance Counters to Model and Optimize the Energy and Performance of Large Scale Scientific Applications on Power-Aware Supercomputers," in *IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW), Chicago, IL*, Chicago, 2016.

[14] W. L. B. a. L. K. John, "Complete System Power Estimation Using Processor Performance Events," *IEEE Transactions on Computers*, vol. vol. 6, no. no. 4, pp. pp. 563-577, April 2012.

[15] C. Jaber, "High-Level soc modeling and performance estimation applied to a multi-core implementation of LTE enodeb physical layer," in *Télécom ParisTech*, 2011.

[16] M. R. C. D. D. F. Johannes Kohl, "A SystemC Based Framework forCycle Accurate Processor Simulationand Parameter Analysis," in *International Federation of Automatic Control*, 2016.

[17] S. M. G. P. a. M. P. F. Fummi, "Native ISS-SystemC integration for the co-simulation of multi-processor SoC," in *Proceedings Design, Automation and Test in Europe Conference and Exhibition*, Paris, 2004.

[18] E. Berg, "A statistical multi-processor cache model," in *IEEE International Symposium on Performance Analysis of Systems and Software*, 206.

[19] S. V. d. Steen, "Analytical Processor Performance and Power Modeling Using Micro-Architecture Independent Characteristics," *IEEE Transactions on Computers*, Vols. vol. 65, no. 12, pp. pp. 3537-3551, 1 Dec. 2016.

[20] G. D. a. H. D. Wilding M, "Efficient Simulation of Formal Processor Models," 1998.

[21] G. W. C. a. C. Hardin.D, "Single-ThreadedFormalProcessorModels:EnablingProofandHigh-SpeedExecution," 1999.

[22] M. S. a. D. F. M. Reichenbach, "Analytical Model for the Optimization of Self-Organizing Image Processing Systems Utilizing Cellular Automata," in *14th IEEE International Symposium on Object/Component/Service-Oriented Real-Time Distributed Computing Workshops*, Newport Beach, 2011 .

[23] C. S. B. R. a. K. W. C. S. Song, "A Simplified and Accurate Model of Power-Performance Efficiency on Emergent GPU Architectures," in *IEEE 27th International Symposium on Parallel and Distributed Processing*, Boston, 2013 .

[24] A. T. M. T. G. D. Kim N.S., "Challenges for Architectural Level Power Modeling," *Graybill R., Melhem R. (eds) Power Aware Computing. Series in Computer Science*, 2002.

[25] D. J. K. C. a. S. H. Jintaek Kang, "Fast Performance Estimation and Design Space Exploration of Manycore-based Neural Processors," in *56th ACM/IEEE Design Automation Conference (DAC)*, Las Vegas, 2019 .

[26] W.-K. C. S. d. J. A. E. R. M. M. D. D. M. J. C. Sanjay Bhansali, "Framework for Instruction-level Tracing and Analysis of Program Executions," in *Proceedings of the 2nd International Conference on Virtual Execution Environments*, 2006.

[27] D. K. a. P. E. P. S. B. Kotsiantis, "Data Preprocessing for Supervised Leaning," *INTERNATIONAL JOURNAL OF COMPUTER SCIENCE* , vol. VOLUME 1, NUMBER 1 2006.

[28] Y. L. a. Y. L. J. Chen, "Predictive model based on decision tree combined multiple regressions," in *13th International Conference on Natural Computation, Fuzzy Systems and Knowledge Discovery (ICNC-FSKD)*, Guilin, 2017.

[29] I. M. a. A. S. S. Pathak, "An Assessment of Decision Tree based Classification and Regression Algorithms," in *3rd International Conference on Inventive Computation Technologies (ICICT)*, Coimbatore, India, 2018.

[30] D. M. N. d. F. Misha Denil, "Narrowing the Gap: Random Forests," in *Proceedings of the 31 st International Conference on Machine*, Beijing, China, 2014.

[31] G. Biau, "Analysis of a Random Forests Model," *Journal of Machine Learning Research* , pp. 1063-1095, 2012.

[32] A. Zheng, Evaluating Machine Learning Models, O'Reilly Media, Inc, 2015.

[33] T. L. L. H. Refaeilzadeh P., Cross-Validation, Springer, Boston, MA, 2009.

[34] R. E. S. Yoav Freund, "A Short Introduction to Boosting," *Journal of Japanese Society for Artificial Intelligence*, vol. 14, pp. 771-780, September, 1999.

[35] L. H. ,. X. B. TU Chengsheng, "AdaBoost typical Algorithm and its application research," in *MATEC Web of Conferences, ICMITE*, 2017.

[36] A.-L. B. a. B. B. Philipp Probst, "Tunability: Importance of Hyperparameters of Machine," *Journal of Machine Learning Research 20*, pp. 1-32, 2019.