The Tribes of Machine Learning and the Realm of Computer Architecture

AYAZ AKRAM, University of California, Davis

JASON LOWE-POWER, University of California, Davis

Machine learning techniques have influenced the field of computer architecture like many other fields. This paper studies how the fundamental machine learning techniques can be applied towards computer architecture problems. We also provide a detailed survey of computer architecture research that employs different machine learning methods. Finally, we present some future opportunities and the outstanding challenges that need to be overcome to exploit full potential of machine learning for computer architecture.

Additional Key Words and Phrases: machine learning, computer architecture

1 INTRODUCTION

Machine learning (ML) refers to the process in which computers learn to make decisions based on the given data set without being explicitly programmed to do so [8]. There are various classifications of the ML algorithms. One of the more insightful classifications has been done by Pedro Domingos in his book *The Master Algorithm* [39]. Domingos presents five fundamental tribes of ML: the symbolists, the connectionists, the evolutionaries, the bayesians and the analogizers. Each of these believe in a different strategy to go through the learning process. These tribes or schools of thought of ML along-with their primary algorithms and origins are shown in Table 1. There are existing proofs that given the enough amount of data, each of these algorithms can fundamentally learn anything. Most of the well known ML techniques/algorithms [1] belong to one of these tribes of ML.

Table 1. Five Tribes of ML (taken from [39])

Tribe	Origins	Master Algorithms
Symbolists	Logic, philosophy	Inverse deduction
Connectionists	Neuroscience	Backpropagation
Evolutionaries	Evolutionary biology	Genetic programming
Bayesians	Statistics	Probabilistic infernce
Analogizers	Psychology	Kernel machines

In this paper, we look at these five school of thoughts of ML and identify how each of them can be fundamentally used to solve different research problems related to computer architecture. ML techniques have already influenced many domains of computer architecture. Figure 1 shows the number of research works using ML each year since 1995. It is observed that most of the work employing ML techniques (approximately 65% of the studied work) is done in last 5-6 years indicating increasing popularity of ML models in computer architecture research. Findings also indicate that Neural Networks are the most used ML technique in computer architecture research as shown in Figure 2.

---

[1]See Appendix A for a detailed summary of many ML techniques referred in this paper.

Authors' addresses: Ayaz Akram, University of California, Davis, yazakram@ucdavis.edu; Jason Lowe-Power, University of California, Davis, jlowepower@ucdavis.edu.
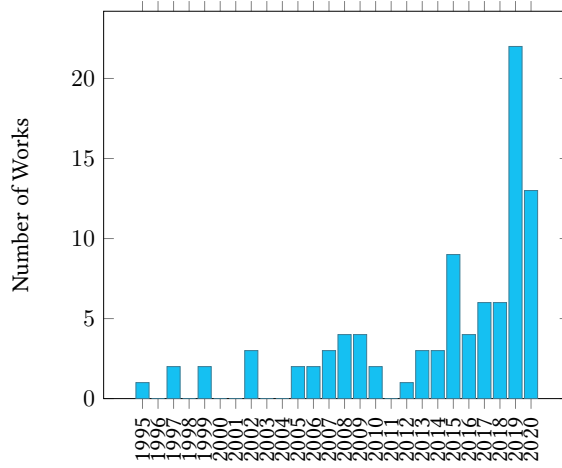
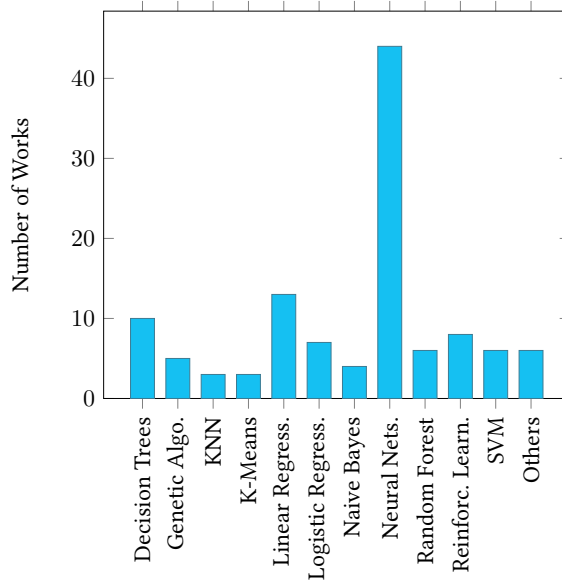Fig. 1. Number of works done in chronological order

Fig. 2. Number of works for each ML Algorithm

We also present an extensive survey of the research works employing these techniques in computer architecture. Most of the insights discussed in this paper are based on a review of more than a hundred papers [2] which use ML for computer architecture.

The specific questions we answer in this paper include:

- What are the fundamental features of different ML algorithms which make them suitable for particular architecture problems compared to the others?

---

[2]Detailed summaries of most of these papers are available in: https://github.com/ayaz91/Literature-Review/blob/main/ML_In_CompArch/ML_CompArch.md

- How has ML impacted computer architecture research so far?
- What are the most important challenges that need to be addressed to fully utilize ML potential in computer architecture research?

2 SURVEY OF USE OF MACHINE LEARNING IN COMPUTER ARCHITECTURE

Table 2. ML Tribes and Examples of Their Use in Computer Architecture

Tribe	Targeted Problem	Example Algorithms	Example Works
Symbolists	Knowledge Composition	Inverse deduction, Decision trees	Microarchitecture [45, 105, 130, 144], Performance Estimation [43, 79], Scheduling [56], Energy [122], instruction scheduling [112, 124]
Connectionists	Credit Assignment	Deep-Neural Networks, Perceptrons	Microarchitecture [27, 37, 69, 76, 84, 113, 158, 173, 185, 193], Performance Estimation [43, 189], scheduling [17, 56, 102, 131, 186, 187], DSE [60, 85], Energy [188], Security [31, 86, 88, 135, 137]
Evolutionaries	Structure Discovery	Genetic Algorithm	Performance estimation [59] , Design Space Exploration [44, 145, 167], instruction scheduling [34, 95]
Bayesians	Uncertainty Reduction	Naive-Bayes, LDA	Microarchitecture [13], DSE [145]
Analogizers	Similarity Discovery	SVM, KNN	Microarchitecture [36, 105], Performance [12, 43, 59], Scheduling [56, 186, 187], Security [137]

In this section, we will discuss how each of the previously mentioned paradigms of ML can be (or have been) applied to the field of computer architecture. Table 2 enlists the five tribes of ML along-with their targeted problem and some examples of the relevant research works. Of course, many architectural problems can be solved by more than one of these families of ML algorithms.

2.1 The Symbolists:

This tribe of ML relies on symbol manipulation to produce intelligent algorithms. The fundamental problem that Symbolists are trying to solve is knowledge composition. Their insight is to use initial knowledge to learn quicker than learning from scratch. More specific examples of this group of ML algorithms include inverse deduction and decision trees. Algorithms belonging to Symbolists seem to be ideal to be used for cases where a cause and effect relationship needs to be established between events. For example, the architecture problems where we need this type of learning include: finding reasons for hardware security flaws and consistency bugs.

The more interpretable nature of these algorithms make them a good candidate to understand the impact of certain design features (input events) on the performance metric of the entire system. As a result, certain parameters can be fine tuned to produce desirable results. For example, this kind of learning can be applied to understand which pipeline structures consume the most amount of energy in a given configuration. However, it might be hard to map these algorithms directly to the hardware (if desired to be used at run-time) due to their symbolic nature.

There exist some examples of usage of these algorithms in computer architecture research. Fern et al. [45] introduced decision tree based branch predictors. Decision trees allowed the proposed branch predictor to be controlled by different processor state features. The relevant features could change at run-time without increasing linearly in size with the addition of new features (compared to table based predictors), providing significant benefits over conventional table based predictors.

Decision tree based models have been used in many cases to understand the impact of different architectural events on systems' performance. Yount et al. [43] compared various machine learning (ML) algorithms with respect to their ability to analyze architectural performance of different workloads and found tree-based ML models to be the most interpretable and almost as accurate as Artificial Neural Networks (ANNs). Jundt et al. [79] used a ML model named Cubist [1], which uses a tree of linear regression models to observe the importance of architectural blocks that have an effect on performance and power of high performance computing applications on Intel's Sandy Bridge and ARMv8 XGene processors. Mariani et al. [115] used Random Forest to estimate HPC (High Performance Computing) applications' performance on cloud systems using hardware independent profiling of applications.

Rahman et al. [144] used decision trees and logistic regression to build framework to identify the best prefetcher configuration for given multithreaded code (in contrast to focus on serial code as in [105]). Hardware prefetcher configuration guided by the presented machine learning framework achieved close to 96% speed-up of optimum configuration speed-up.

Moeng and Melhem [122] used decision trees (implemented in hardware) to propose a DVFS (dynamic voltage and frequency scaling) policy that will be able to predict clock frequency resulting into the least amount of energy consumption in a multicore processor. They used simple measurement metrics like cycles, user instructions, total instructions, L1 accesses and misses, L2 accesses, misses and stalls for a DVFS interval during execution as inputs to the decision trees.

2.2 The Connectionists:

The connectionists rely on brain structure to develop their learning algorithms and try to learn connections between different building blocks (neurons) of the brain. The main problem they are trying to solve is of credit assignment i.e. figure out which connections are responsible for errors and what should be the actual strength of those connections. The most common example of this class of ML algorithms is (deep) neural networks. The algorithms belonging to the Connectionists are good at learning complex patterns specially at run time (as evident by many microarchitecture design examples). This type of learning can be useful in many microarchitectural predictive structures (like branch predictors, cache prefetchers and value predictors) which try to forecast an event based on similar events of the past, where different past events might have different weights in determining if a certain event will happen in the future (the problem of credit assignment). In contrast to symbolists, the algorithms belonging to this group might not need any initial knowledge, however they are not very interpretable. So, they might not be very useful to understand the importance of different architectural events/components in overall performance/output for static analysis. This kind of algorithms have found uses in the architecture research, specially simple neural networks like perceptrons [19] (owing to their simple structure and being more amenable to be implemented in the hardware).

Calder et al. [27] introduced the use of neural networks for static branch prediction in the late 1990's. One of the earliest works to use ML for dynamic branch prediction was done by Vinton and Iridon [179]. They used neural networks with Learning Vector Quantization (LVQ) [94] as a learning algorithm for neural networks and were able to achieve around 3% improvement in misprediction rate compared to conventional table based branch predictors. Later, Egan et al. [42] and Wang and Chen [185] also used LVQ for branch prediction. The complex hardware implementation

of LVQ due to computations involving floating point numbers, could significantly increase the latency of the predictor [76]. Jimenez et al. [76], working independently, also used neural network based components, perceptrons [19], to perform dynamic branch prediction. In their work, each single branch is allocated a perceptron. The inputs to the perceptron are the weighted bits of the "global branch history shift register", and the output is the decision about branch direction. One big advantage of perceptron predictor is the fact that the size of perceptron grows linearly with the branch history size (input to the perceptron) in contrast the size of pattern history table (PHT) in PHT based branch predictors which grows exponentially with the size of the branch history. Therefore, within same hardware budget, perceptron based predictor is able to get a benefit from longer branch history register. One of the big problems with the use of perceptrons is their inability to learn linear separability [3]. This was later resolved by Jimenez [71] using a set of piecewise linear functions to predict the outcomes for a single branch. These linear functions refer to a distinct historical path that lead to the particular branch instruction. Graphically, all these functions, when combined together, form a surface.

Since the introduction of neural network based branch predictors, there has been a lot of research done to optimize the design of these predictors. Different analog versions have been proposed to improve speed and power consumption of the neural network based branch predictors [9, 73, 92, 151, 166]. Perceptron based predictors have also been combined with other perceptron or non-neural network based predictors to achieve better accuracy overall: [42, 67, 72, 98, 123, 149, 165, 170]. Different optimizations/modifications to neural network branch predictors including use of different types of neural networks have been explored: [26, 50, 52, 54, 58, 69, 70, 90, 132, 156, 157, 160, 174]. These optimizations improve performance of the predictor and save power and hardware cost. Similar concepts are also used in development of other branch predictor related structures like confidence estimators and indirect branch predictors: [6, 38, 68, 75, 82]. The statistical corrector predictor used in TAGE-SC_L [158] branch predictor [4] is also a perceptron based predictor. This statistical corrector predictor regresses TAGE's prediction if it statistically mispredicts in similar situations. Mao et al. [113], recently, applied deep learning (easy to train Deep Belief Networks (DBN) [57]) to the problem of branch prediction.

The popularity of perceptrons in branch prediction has also affected the design of other microarchitecture structures. For example, Wang and Luo [182] proposed perceptron based data cache prefetching. The proposed prefetcher is a two level prefetcher which uses conventional table-based prefetcher at the first level. At the second level a perceptron is used to reduce unnecessary prefetches by relying on memory access patterns. Peled et al. [139] used neural networks to capture semantic locality of programs. Memory access streams alongwith a machine state is used to train neural network at run-time which predicts the future memory accesses. Evaluation of the proposed neural prefetcher using SPEC2006 [163], Graph500 [127] benchmarks and other hand-written kernels indicated an average speed-up of 22% on SPEC2006 benchmarks and 5x on other kernels. However, importantly, Peled et al. [139] also performed a feasibility analysis of the proposed prefetcher which shows that the benefits of neural prefetcher are outweighed by other factors like learning overhead, power and area efficiency. Teran et al. [173] applied perceptron learning algorithm (not actual perceptrons) to predict reuse of cache bkocks using features like addresses of recent memory instructions and portions of address of current block. Verma [177] extended the work of Teran et al. [173] and proposed CARP (coherence-aware reuse prediction). CARP uses cache coherence information as an additional feature in the alogrithm of Teran et al. [173]. Seng and Hamerly

---

[3]A boolean function is "linearly separable" if all false instances of the function can be separated from its all true instances using a hyperplane [121]. As an example XOR is linearly inseparable and AND is linearly separable.
[4]won the 2014 branch predictor championship and combines TAGE branch predictor with a loop predictor and a statistical corrector predictor

[155] did one of the earliest works to present perceptron based register value predictor, where each perceptron is fed with the global history of recently committed instructions. Later, Black and Franklin [18] proposed perceptron based confidence estimator for a value predictor. Nemirovsky et al. [131] proposed a neural network based scheduler design for heterogeneous processors.

These algorithms have also been used to build performance/power models. For example, Khan et al. [85] proposed feed-forward neural network based predictive modeling technique to do design space exploration for chip multiprocessors. Wu et al. [189] proposed a neural network based GPU performance and power prediction model to solve the problem of slow simulation speed of simulators to study performance/power of GPUs. The proposed model estimates performance and power consumption of applications with changes in GPU configurations.

Dai et al. [37] exploited deep learning techniques to propose *Block2Vec* (which is inspired by Word2Vec [118] used in word embeding), which can find out correlations among blocks in storage systems. Such information can be used to predict the next block accesses and used to make prefetching decisions. Khakhaeng and Chantrapornchai [84] used perceptron based neural network to build model to predict ideal cache block size. Neural network is trained using features from address traces of benchmarks. The particular features used for training include: cache misses and size and frequency adjoining addresses which reflects temporal and spatial locality of the program.

Neural networks have also been used (both offline and online) to discover the optimal schedule of workloads or manage other aspects of shared hardware resources. For example, Bitirgen et al.. [17] implemented Artificial Neural Network (ANN) in the hardware for the management of shared resources in multiprocessor systems. Each ANN (edge weights are multiplexed at the run time to achieve virtual ANNs using one hardware ANN) acts as a performance model and takes available resources and other factors describing the program behavior (e.g. read/write hits and misses in data cache and the portion of dirty cache ways allocated to the applications) as inputs and outputs the information which helps to decide which resource distribution would result in the best overall performance. Li et al. [102] used ANNs to predict the performance of parallel tasks at run-time to do an optimal scheduling. Nemirovsky et al. [131] proposed an ANN based approach to perform scheduling on heterogeneous processors, which increases throughput by 25-31% compared to a *RoundRobin* scheduler on an ARM big.Little system [55]. The proposed methodology relies on ANNs to predict thread's performance for a particular scheduling interval on different hardware core types and the scheduler picks the schedule of threads that would maximize the system performance.

Other examples of the use of neural networks include the work of Ipek et al. [60] for design space exloration and the work of Chiappetta et al. [31] to detect cache based side channel attacks (e.g. Flush+Reload [192]) relying on the hardware performance counter values.

2.3 The Evolutionaries:

This class of algorithms are based on the evolutionary process of nature and rely on learning structures instead of learning parameters. These algorithms keep on evolving and adapting to unspecified surroundings. The most obvious example are genetic algorithms, which potentially can be used for searching the best design in a large design space. Another problem these algorithms are a natural fit to solve is of an attacker which evolves to different versions once a preventive technique is deployed (e.g. different ML based side channel attack detection and mitigation solutions). These algorithms can help in building preventive techniques which can evolve with the evolution of the attacker. General purpose computer systems might be used for a diverse set of applications after being deployed. It seems reasonable to make these systems able to evolve their configuration at runtime depending on their use. Genetic algorithms seem to be the most suitable algorithms to learn how to adapt to the best possible system configuration depending on the workloads being executed.

There exist only a limited number of examples of the utilization of these algorithms by computer architects. For example, Joel et al. [44] used genetic algorithms to design better branch predictors and Jimenez et al. [74] took help of genetic algorithms to introduce a pseudo-LRU (least recently used) insertion and promotion mechanism for cache blocks in last level caches, which could result in 5% speedup compared to traditional LRU (least recently used) algorithm using much less overhead. The performance of the proposed mechanism matched other contemporary techniques like DRRIP (dynamic rereference interval prediction [63]) and PDP (protecting distance policy [41]) with less overhead. Mukundan and Martinez [125] used genetic algorithms to propose MORSE (Multi-objective Reconfigurable Self-optimizing Memory Scheduler) extending Ipek et al's work [61] which used reinforcement learning for memory scheduling. MORSE can target optimization of different metrics like performance, energy and throughput.

2.4 The Bayesians:

The algorithms belonging to the Bayesians are usually concerned with incorporating new evidence into previous beliefs. The main problem they try to solve is of reducing uncertainty. They are specially good at predicting events when there is a known probability of occurrence of particular events. Naive Bayes and Linear Discriminant Analysis (LDA) are a couple of examples of this group of algorithms. These algorithms have a high potential to be used by the architects due to their inherent characterstics, as a lot of architecture problem solutions rely on the knowledge of existence of certain events. For instance, a bayesian based model to detect hardware side channel attacks can rely on the fact that the probability of system under no attack would be very high normally. Similarly, a model to predict memory events can depend on the average probability of an instruction being a memory operation. Beckman and Sanchez [13] recognized that the problem of design of cache replacement policies have to deal with uncertainty as the time when the candidate cache blocks will be accessed is not known. Thus, bayesians have the potential to learn in such environments and develop smart cache replacement policies.

Overall, Bayesians also have not found a lot of use in the computer architecture community as our survey suggests. Jung and Pedram [80] used Bayesian classification to tune voltage and frequency settings to reduce system's energy consumption in a multicore processor. Reagen et al. [145] used Bayesian optimization [141] to explore design space for a Deep Neural Network hardware accelerator. Different design parameters studied in this work, like neurons/layer for DNN, L2 regularization for DNN, learning rate of DNN, loop parallelism in hardware, and hardware pipelining, have "complex interactions" among them which the bayesian optimization tries to learn.

2.5 The Analogizers:

Analogizers believe in learning things from other examples. The primary problem they target is finding similarities between different situations. Few examples of algorithms belonging to this domain include: support vector machines (SVM), k-means clustering and k-nearest neighbor (KNN). Some of these algorithms (like k-means clustering and KNN) are used in un-supervised learning. Algorithms from this group can be good candidates to be applied in cutting down the search space for design space exploration especially when configuration space is large and there exists multiple designs with marginal differences. These algorithms can help to focus on designs which are sufficiently different from a baseline. Similarly, they can help in detecting similarities at the granularity of programs or at the granularity of instructions to guide better predictions at the run-time by different microarchitectural structures.

Analogizers have been pretty successful in the domain of computer architecture. For example, Culpepper and Gondree [36] used support vector machines (SVM) to improve accuracy of branch prediction. This SVM based branch predictor performed better compared to fast-path based predictor

and gshare predictor at high hardware budgets [36]. Sherwood et al. [159] used K-means clustering to form groups/clusters of similar basic block vectors (BBV [5]). These groups then act as a few representative portions of the entire program and are known as *Simpoints*, which can be used to approximate the performance/power for the entire program. Hoste et al. [59] detected similarity among programs using data from various microarchitecture-independent statistics (like instruction classes, instruction-level parallelism and register traffic, branch predictability and working set sizes) to predict performance of programs similar to reference benchmarks with known performance on specific microarchitecture. They used principal component analysis for this purpose alongwith other ML techniques.

Walker et al. [181] relied on heirarchical cluster analysis [25] to form groups of various performance monitoring counters which are then used to estimate power consumption for mobile and embedded devices and showed that their technique leads to lower percentage error compared to many other power estimation techniques [142, 148, 150, 180].

Baldini et al. [12] trained two binary classifiers Nearest Neighbor with Generalized exemplars (NNGE) and Support Vector Machine (SVMs) to predict possible GPU speed-up given a run of parallel code on CPU (using OpenMP implementation). Wang and Ipek [184] proposed an online data clustering based technique to reduce energy of data transfers in memory by reducing the number of ones in the data.

2.6 Reinforcement Learning: The Sixth Tribe [6]

Reinforcement learning refers to the process in which an agent learns to take actions in a given environment through independent exploration of different possible actions and choosing the ones that increase the overall reward. Reinforcement learning does not require any prior data to learn to take the right action, rather it learns to do this on the fly. This makes it a good fit to be used for computer architecture problems where significant former knowledge or data is not available and the right action can only be learned dynamically. For example, hardware optimizations or hardware based resource management techniques which are totally transparent to the software can rely on reinforcement learning to decide the best action to take for the overall efficiency of the system.

One of the earliest examples of the use of reinforcement learning in computer architecture is the work of Ipek et al [61]. Ipek et al [61] introduced reinforcement learning based DRAM scheduling for an increased utilization of memory bandwidth. The DRAM scheduler which acts as a reinforcement learning agent utilizes system state defined by different factors like number of read/write requests residing in the transaction queue. The actions that the agent can take include all normal commands of DRAM controller like read, write, pre-charge and activate. Ipek et al [61] implemented a five stage hardware pipeline to calculate q-values [169] associated with the credit assignment which determine the eventual benefit for an action given the current state.

Peled et al [138] used reinforcement learning to approximate program semantic locality, which was later used to anticipate data access patterns to improve prefetching decisions. Accesses are considered to have semantic locality if there exists a relationship between them via a series of actions. A subset of different attributes (e.g. program counter, accesses history, branch history, status of registers, types and offsets of objects in program data structures and types of reference operations) is used to represent the current context a program.

Juan and Marculescu [78] proposed a reinforcement learning based DVFS technique which divides the central agent in many "distributed" agents and uses a supervisor for coordination

---

[5]BBV contains the frequency of occurrence of all basic blocks during an interval of execution
[6]We refer to reinforcement learning as the sixth tribe of ML as it is one of the most important algorithms belonging to the class of self learning algorithms. Reinforcement learning does not belong to the original ML tribes taxonomy of Pedro Domingos [39].

among other agents to maximize performance under given power budget. Similarly, Ye and Xu [194] proposed a reinforcement learning based dynamic power management (DPM) technique for multi-core processors.

2.7 Regression Techniques – Statistics Meets Machine Learning [7]

Regression techniques from applied statistics have been largely borrowed by machine learning. A detailed discussion on different regression techniques used in machine learning is provided in the Appendix section. In this section, we take a look at how these regression techniques have influenced the field of computer architecture.

Performance and power estimation of different workloads on a particular architecture is a critical step to design new architectures or to have a better understanding of the already existing architectures. There are numerous examples of research works where regression techniques are used to estimate performance or power consumption. Performance/power estimation of applications on a particular architecture is mostly done by simulation tools and analytical models. Regression techniques have also been used to build new and accurate models using empirical data. Sometimes they are also used to increase the accuracy of simulation techniques. For example, Lee et al. [99] proposed regression analysis based calibration methodology called *PowerTrain* to improve the accuracy of McPAT [103]. McPAT [103] is a well-known power estimation simulator, but it is shown to have various inaccuracies due to different sources like un-modeled architectural components, discrepancy between the modeled and actual hardware components and vague configurational parameters [190]. The proposed calibration methodology called *PowerTrain* uses power measurements on real hardware to train McPAT using regression analysis. Reddy et al. [146] studied correlation among gem5 [16, 110] statistics and hardware performance monitoring counters to build a model in gem5 to estimate power consumption of ARM Cortex-A15 processor. This work uses the same model built by Walker et al. [181] but does not include all performance monitoring events as some gem5 equivalents would not be available. The used events in ML model are cycle counts, speculative instructions, L2 cache accesses, L1 instruction cache accesses and memory bus reads. Reddy et al. [146] show that the differences between statistics of the simulator (gem5) and those of the real hardware only affect the estimated power consumption by approximately 10%.

There are also examples [97, 142] of use of regression techniques to estimate performance of multi-core processors from single core processors. Lee et al. [97] used spline-based regression to build model for multiprocessor performance estimation from uniprocessor models to mitigate the problems associated with cycle accurate simulation of multiprocessors. Pricopi et al. [142] used regression analysis to propose an analytical model to estimate performance and power for heterogeneous multi-core processors (HMPs). During an application run, a cpi (cycles per instruction) stack is built for the application using different micro-architectural events that can impact the execution time of the application. Relying on some compiler analysis results alongwith the cpi stack, performance and power can be estimated for other cores in an HMP system. Experiments performed with an ARM big.Little system indicate an intra-core prediction error of below 15% and an inter-core prediction error of below 17%.

Regression techniques are also used in heterogeneous systems for cross platform performance estimation. Examples include [199], [20], [133], and [10]. Zheng et al. [199] used Lasso Linear Regression and Constrained Locally Sparse Linear Regression (CLSLR) to explore correlation among performance of same programs on different platforms to perform cross-platform performance prediction. The test case used in their work is prediction of an ARM ISA platform based on performance on Intel and AMD x86 real host systems. The authors extended this work in [198]

---

[7]Regression techniques are not a part of the original ML tribes taxonomy of Pedro Domingos [39]

by proposing *LACross* framework which applies similar methodology to predict performance and power at fine granularity of phases. *LACross* is shown to have an average error less than 2% in entire program's performance estimation, in contrast to more than 5% error in [198] for SD-VBS benchmark suite [176]. Boran et al. [20] followed Pricopi et al.'s [142] work and used regression techniques to estimate execution cycle count of a particular ISA core based on the performance statistics of another ISA core. This model is used to dynamically schedule programs in a heterogeneous-ISA multi-core system. ARMv8 and x86-64 based multi-core system is used to validate the model. Although this model shows above 98% accuracy to estimate performance on any particular ISA core, the inter-core performance estimation has high error (23% for estimation from ARM to x86 and 54% for estimation from x86 to ARM). O'Neal et al. [133] used various linear/non-linear regression algorithms to propose GPU performance predicting model focusing on "pre-silicon design" of GPUs using DirectX 3D workloads. Ardalani et al. [10] proposed *XAPP* (*Cross Architecture Performance Prediction*) technique to estimate GPU performance from CPU code using regression and bootstrap aggregating (discussed in the Appendix section). Different features associated with the program behavior are used to train machine learning models. These features include some basic features related to ILP (instruction level parallelism), floating point operations and memory operations and also some advanced features like shared memory bank utilization, memory coalescing and branch divergence.

Regression techniques have also been used in design space exploration (e.g. [66, 96]). Lee et al. [96] used regression modeling for fast design space exploration to avoid expensive cycle accurate simulations. Jia et al. [66] proposed *StarGazer*, an automated framework which uses a stepwise regression algorithm to explore GPU design space by using only few samples out of all possible design points and estimates the performance of design points with only 1.1% error on average.

Some regression techniques have also been used in the domain of hardware security, specifically to detect the malware or micro-architectural side channel attacks. For instance, Ozosoy et al. [135] used neural network and logistic regression for detection of malware using hardware architectural features (like memory accesses and instruction mixes). They evaluated the FPGA based implementations of both ML models (logistic regression and neural networks). Khasawneh et al. [86] used neural networks and logistic regression (LR) based hardware malware detectors to prove the possibility of evading malware detection. Availability of malware detector training data makes it possible for attackers to reverse engineer detectors and potentially modify malware to evade their detection. Khasawneh et al. [86] also proposed randomization based technique to attain resilient malware detection and avoid reverse engineering. Many other works which utilize regression or other ML techniques for micro-architectural side channel attack detection are referred in [7].

3 SUMMARY TABLE

Table 3 provides a summary of the previously discussed literature survey at a finer granularity of individual ML techniques. Each column in this table refer to a broader category (or field) of computer architecture that has relied on ML.

Table 3. Summary of the Literature Survey on Machine Learning in Computer Architecture

Technique	Microarchitecture	Power/Perform. Estimation	Thread Scheduling	Design Space Exploration	Energy Improvement	Instruction Scheduling	Hardware Security
Decision Trees	[45, 105, 130, 144]	[29, 43, 79]	[56, 143]	[122, 154, 200]		[112, 124]	[128, 129]
Genetic Algo.	[44]	[59]		[145, 167]		[34, 95]	
K-Nearest Neighbor	[105]	[12, 29]	[56]				
K-Means		[79, 159, 189]			[111, 184]		
Linear/Non-Linear Regression		[5, 10, 20, 43, 99, 133, 134, 136, 142, 146, 181, 198, 199]	[120, 143]	[66, 96]	[191]		[128, 129]
Logistic Regression	[105, 144]	[29]		[33, 109]			[86–88, 129, 135, 137],
Bayes Theory	[130]			[85, 145]	[80]		[137]
Neural Network	[14, 15, 27, 37, 40, 51, 69, 76, 84, 107, 113, 114, 140, 158, 161, 162, 171–173, 185, 193, 195, 196]	[29, 43, 48, 65, 117, 134, 147, 189]	[17, 53, 56, 102, 131, 186, 187]	[22, 60, 83, 85, 104, 106, 109, 154]	[188]	[62]	[31, 86, 88, 119, 129, 135, 137]
PCA		[59]					
Random Forest		[11, 115, 134, 136]	[143]	[154]			
Reinforcement Learning	[61, 91, 138, 152, 183, 197]		[47]		[46, 78, 194]	[116]	
SVM	[36, 105]	[12, 43]	[56, 186, 187]	[154]			[128, 129, 137]
Others	[71, 130, 178]	[43, 181]	[120]	[145]	[101]	[4]	[137]

Note: The examples of sub-problems for each column in this table are following: **Microarchitecture:** branch prediction, cache replacement, cache reuse prediction, value prediction, memory scheduling, prefetching, network-on-chip design. **Power/Perform. Estimation:** power/performance estimation for single or multi-cores, gpus, cross-platform (e.g. one ISA to other or cpu to gpu), simulation augmentation. **Thread Scheduling:** management of shared hardware resources, thread scheduling in heterogeneous systems. **Design Space Exploration:** design space exploration of single or multi-cores, gpus, accelerators, network-on-chips. **Energy Improvements:** energy aware thread assignment or micro-architecture design, dynamic voltage and frequency scaling. **Instruction Scheduling:** static/dynamic instruction scheduling. **Hardware Security:** micro-architectural side-channel attack detection and evasion.

4 OPPORTUNITIES AND CHALLENGES:

This section presents some opportunities to use ML for architecture that have not been explored in their full potential if explored at all. Moreover, we also enlist a number of challenges that need to be tackled to be able to fully exploit ML for architecture research.

4.1 Opportunities:

- There exists an opportunity to borrow ideas from ML instead of using ML algorithms in their intact form. Since, implementing ML algorithms in the hardware can be costly, computer architects can rely on ML algorithms to optimize the classical learning techniques used in computer systems. For instance, the work done by Teran et al. [173] to apply perceptron learning algorithm to predict reuse of cache blocks is a good example of this.
- ML based system components can be used in synergy with traditional components. A good example of this collaboration is the work of Wang and Luo [182] to perform data cache prefetching discussed in the previous section.
- ML techniques can potentially be used to build customizable computer systems which will learn to change their behavior at different granularities. Take an example of an 'A' cache eviction policy which works better for workload type 'X' and another policy 'B' which works better for workload type 'Y'. Since, ML makes learning easier, same processor can learn to adapt to using policy 'A' or 'B' depending on the workloads run on it.
- Computer architects have relied on heuristics to design different policies. One possiblity is to start using ML in place of these heuristics. Machine Learning can also help us to come up with new and better heuristics that are hard to ascertain otherwise.
- For all the decisions taken inside a computer system, multiple policies are used. If single (or a small number of) ML algorithm(s) can learn all of these policies, the design of systems can be significantly simplified.
- As we observed earlier, performance and power estimation of existing or new architectures is an area which can be highly influenced by ML. Architects have historically used simulation tools or mathematical models for this purpose. Future architectural innovations are expected to come from optimizations across the entire software/hardware computing stack [100]. Current simulation techniques are not designed with this kind of use case in mind. Therefore, ML techniques can come to rescue and enable building simulation tools of the future as traditional cycle level modeling might be too slow for cross-stack studies. Not only that ML techniques can enable building new tools, they can help in improving the accuracy of current simulation tools (or performance/power models) which are the primary enabler of computer architecture research at the moment (for example the work by Renda et al. [147]).
- ML can specially prove useful to build systems/accelerators for ML.
- Design space explorations (which are extensively done in computer architecture research) should use more of ML techniques (especially the ones belonging to evolutionaries and analogizers). Techniques like genetic algorithms can help to evolve the design space itself as well, thus enabling the evaluation of significant configurations which might remain hidden otherwise. Since, all of this exploration is done statically (is part of pre-silicon design), there will be no issues of the cost of the algorithm here.
- Computer architecture research often relies on combination of multiple (independently functioning) structures to optimize the system. For example, tournament branch predictors use multiple branch predictors and pick one of them for a given prediction. Similar tasks can be relegated to ML to learn the best of different independent techniques.

- Computer architects often run a number of benchmarks/workloads on a multitude of hardware configurations (and generate multiple statistics) when studying new or existing ideas. This leads to huge amounts of data which researchers often study in a limited fashion on ad hoc basis. In contrast, if ML based analysis is performed on this data, we might be able to see the true potential of that data and its ability to help us discover new findings.

4.2 Challenges:

- The resources needed by these ML techniques when they are implemented in hardware can be significant. Specially, if these techniques need to be used for microarchitectural decisions at a finer granularity (at the program phase level), the hardware implementations become more critical. There is a need to develop more optimized hardware implementations of these algorithms to make them more feasible to be applied towards architectural problems/design.
- ML algorithms primarily rely on huge data sets for learning. In comparison to other fields, there does not exist standardized data sets in architecture domain to be used by ML algorithms.
- Although there exist a number of research works which take help of ML algorithms for computer architecture problems, the use of a particular algorithm is not justified by the authors mostly. As we observed in this paper, there are ML algorithms which naturally fit certain problems. We think it will be sensible to emphasize the use of appropriate algorithms for a given problem, which can help in ensuring the robustness of the proposed ideas.
- ML methods have found more use in branch prediction like problems where the decision to be made is a binary decision (taken/not-taken) as opposed to the problems like memory prefetching where the output has to be a complete memory address (which increases the complexity of the ML model).
- Data selection, cleaning, and pre-processing and then interpreting results is a challenge.
- Formulation of architecture problems as machine learning problem is a general challenge.
- As we pointed out in the opportunities sub-section that ML can help improving the accuracy of simulation tools, that opportunity comes with a challenge of building new tools to embed ML algorithms into simulation techniques. Specifically, the question of how to make the simulation tools talk to ML algorithms needs to be addressed.
- There exist a big disparity between the time that ML algorithms take to make a prediction and the time that often microarchitectural structures take for an estimation (known as time scale gap). This needs to be solved before practical ML based microarchitectural solutions can be designed. An example where this problem has been addressed is the design of branch predictors i.e. the proposal of analog versions of neural network based branch predictors (for example: [9, 73]) to improve latency of prediction.
- It is also not clear how (and if) the ML based architecture design will scale. Traditionally, in computer architecture, many microarchitectural/architectural techniques work the same way irrespective of the workloads used or size of the processor. For heuristics based designs, it is easier for humans to argue about their scalability. However, this might not be true for ML based design (specially if it is not very interpretable). Thus, the question arises if the optimal machine learning based design for a specific problem at hand will work if the workloads change significantly or the hardware resources scale to a bigger budget.
- As ML is finding more and more uses in the real world, many security challenges have been raised. Before computer architecture design can be fully influenced by ML, the attacks (like adversarial ML attacks) possible on ML techniques need to be addressed in the context of hardware design. Moreover, these machine learning implementations might lead to new side-channels and computer architects will need to be cognizant of this to avoid major security vulnerabilities in the future.

5 CONCLUSION

This paper surveys a wide set of representative work in computer architecture which utilize ML techniques. We identify the fundamental properties of different classes of Machine Learning and their relation with the problems in the domain of computer architecture. It is observed that the usage of ML techniques is on the rise and future computer architecture research can further leverage the exciting developments from the field of Machine Learning. However, a number of challenges need to be addressed to fully exploit the ML potential for computer architecture research.

REFERENCES

[1] *An Overview of Cubist*. Retrieved from https://www.rulequest.com/cubist-win.html [Online; accessed 1-June-2017].

[2] Decision tree learning, 2018. Retrieved April 4, 2018 from https://en.wikipedia.org/wiki/Decision_tree_learning.

[3] Random Forest, 2018. Retrieved April 4, 2018 from https://en.wikipedia.org/wiki/Random_forest.

[4] AGAKOV, F., BONILLA, E., CAVAZOS, J., FRANKE, B., FURSIN, G., O'BOYLE, M. F., THOMSON, J., TOUSSAINT, M., AND WILLIAMS, C. K. Using machine learning to focus iterative optimization. In *Proceedings of the International Symposium on Code Generation and Optimization* (2006), IEEE Computer Society, pp. 295–305.

[5] AGARWAL, N., JAIN, T., AND ZAHRAN, M. Performance prediction for multi-threaded applications. In *International Workshop on AI-assisted Design for Architecture (AIDArc), held in conjunction with ISCA* (2019).

[6] AKKARY, H., SRINIVASAN, S. T., KOLTUR, R., PATIL, Y., AND REFAAI, W. Perceptron-based branch confidence estimation. In *Software, IEE Proceedings-* (2004), IEEE, pp. 265–265.

[7] AKRAM, A., MUSHTAQ, M., BHATTI, M. K., LAPOTRE, V., AND GOGNIAT, G. Meet the sherlock holmes' of side channel leakage: A survey of cache sca detection techniques. *IEEE Access 8* (2020), 70836–70860.

[8] ALPAYDIN, E. *Introduction to machine learning*. MIT press, 2014.

[9] AMANT, R. S., JIMÉNEZ, D. A., AND BURGER, D. Mixed-signal approximate computation: A neural predictor case study. *IEEE micro 29*, 1 (2009), 104–115.

[10] ARDALANI, N., LESTOURGEON, C., SANKARALINGAM, K., AND ZHU, X. Cross-architecture performance prediction (xapp) using cpu code to predict gpu performance. In *Microarchitecture (MICRO), 2015 48th Annual IEEE/ACM International Symposium on* (2015), IEEE, pp. 725–737.

[11] ARDALANI, N., THAKKER, U., ALBARGHOUTHI, A., AND SANKARALINGAM, K. A static analysis-based cross-architecture performance prediction using machine learning. *arXiv preprint arXiv:1906.07840* (2019).

[12] BALDINI, I., FINK, S. J., AND ALTMAN, E. Predicting gpu performance from cpu runs using machine learning. In *Computer Architecture and High Performance Computing (SBAC-PAD), 2014 IEEE 26th International Symposium on* (2014), IEEE, pp. 254–261.

[13] BECKMANN, N., AND SANCHEZ, D. Maximizing cache performance under uncertainty. In *2017 IEEE International Symposium on High Performance Computer Architecture (HPCA)* (2017), IEEE, pp. 109–120.

[14] BHARDWAJ, A., AND JANARDHAN, V. Pecc: Prediction-error correcting cache. In *Workshop on ML for Systems at NeurIPS* (2018).

[15] BHATIA, E., CHACON, G., PUGSLEY, S., TERAN, E., GRATZ, P. V., AND JIMÉNEZ, D. A. Perceptron-based prefetch filtering. In *2019 ACM/IEEE 46th Annual International Symposium on Computer Architecture (ISCA)* (2019), IEEE, pp. 1–13.

[16] BINKERT, N., BECKMANN, B., BLACK, G., REINHARDT, S. K., SAIDI, A., BASU, A., HESTNESS, J., HOWER, D. R., KRISHNA, T., SARDASHTI, S., ET AL. The gem5 Simulator. *ACM SIGARCH Comp. Arch. News 39*, 2 (May 2011), 1–7.

[17] BITIRGEN, R., IPEK, E., AND MARTINEZ, J. F. Coordinated management of multiple interacting resources in chip multiprocessors: A machine learning approach. In *Proceedings of the 41st annual IEEE/ACM International Symposium on Microarchitecture* (2008), IEEE Computer Society, pp. 318–329.

[18] BLACK, M., AND FRANKLIN, M. Neural confidence estimation for more accurate value prediction. In *Proceedings of the 12th international conference on High Performance Computing* (2005), Springer-Verlag, pp. 376–385.

[19] BLOCK, H.-D. The perceptron: A model for brain functioning. i. *Reviews of Modern Physics 34*, 1 (1962), 123.

[20] BORAN, N. K., MEGHWAL, R. P., SHARMA, K., KUMAR, B., AND SINGH, V. Performance modelling of heterogeneous isa multicore architectures. In *East-West Design & Test Symposium (EWDTS), 2016 IEEE* (2016), IEEE, pp. 1–4.

[21] BOSER, B. E., GUYON, I. M., AND VAPNIK, V. N. A training algorithm for optimal margin classifiers. In *Proceedings of the fifth annual workshop on Computational learning theory* (1992), ACM, pp. 144–152.

[22] BRAUN, P., AND LITZ, H. Understanding memory access patterns for prefetching. In *International Workshop on AI-assisted Design for Architecture (AIDArc), held in conjunction with ISCA* (2019).

[23] BREIMAN, L. Random forests. *Machine learning 45*, 1 (2001), 5–32.

[24] BREIMAN, L. *Classification and regression trees*. Routledge, 2017.

[25] BRIDGES JR, C. C. Hierarchical cluster analysis. *Psychological reports 18*, 3 (1966), 851–854.

[26] CADENAS, O., MEGSON, G., AND JONES, D. A new organization for a perceptron-based branch predictor and its fpga implementation. In *VLSI, 2005. Proceedings. IEEE Computer Society Annual Symposium on* (2005), IEEE, pp. 305–306.

[27] CALDER, B., GRUNWALD, D., JONES, M., LINDSAY, D., MARTIN, J., MOZER, M., AND ZORN, B. Evidence-based static branch prediction using machine learning. *ACM Transactions on Programming Languages and Systems (TOPLAS) 19*, 1 (1997), 188–222.

[28] CARUANA, R., AND NICULESCU-MIZIL, A. An empirical comparison of supervised learning algorithms. In *Proceedings of the 23rd international conference on Machine learning* (2006), ACM, pp. 161–168.

[29] CARVALHO, P., CLUA, E., PAES, A., BENTES, C., LOPES, B., AND DRUMMOND, L. M. D. A. Using machine learning techniques to analyze the performance of concurrent kernel execution on gpus. *Future Generation Computer Systems 113* (2020), 528–540.

[30] CHANDOLA, V., BANERJEE, A., AND KUMAR, V. Anomaly detection: A survey. *ACM computing surveys (CSUR) 41*, 3 (2009), 15.

[31] CHIAPPETTA, M., SAVAS, E., AND YILMAZ, C. Real time detection of cache-based side-channel attacks using hardware performance counters. *Applied Soft Computing 49* (2016), 1162–1174.

[32] CHRISTOPHER, M. B. *PATTERN RECOGNITION AND MACHINE LEARNING.* Springer-Verlag New York, 2016.

[33] COCHRAN, R., HANKENDI, C., COSKUN, A. K., AND REDA, S. Pack & cap: adaptive dvfs and thread packing under power caps. In *Proceedings of the 44th annual IEEE/ACM international symposium on microarchitecture* (2011), ACM, pp. 175–185.

[34] COOPER, K. D., SCHIELKE, P. J., AND SUBRAMANIAN, D. Optimizing for reduced code space using genetic algorithms. In *ACM SIGPLAN Notices* (1999), vol. 34, ACM, pp. 1–9.

[35] COVER, T., AND HART, P. Nearest neighbor pattern classification. *IEEE transactions on information theory 13*, 1 (1967), 21–27.

[36] CULPEPPER, B. J., AND GONDREE, M. Svms for improved branch prediction. *University of California, UCDavis, USA, ECS201A Technical Report* (2005).

[37] DAI, D., BAO, F. S., ZHOU, J., AND CHEN, Y. Block2vec: A deep learning strategy on mining block correlations in storage systems. In *Parallel Processing Workshops (ICPPW), 2016 45th International Conference on* (2016), IEEE, pp. 230–239.

[38] DESMET, V., EECKHOUT, L., AND DE BOSSCHERE, K. Improved composite confidence mechanisms for a perceptron branch predictor. *Journal of Systems Architecture 52*, 3 (2006), 143–151.

[39] DOMINGOS, P. *The master algorithm: How the quest for the ultimate learning machine will remake our world.* Basic Books, 2015.

[40] DOUDALI, T. D., BLAGODUROV, S., VISHNU, A., GURUMURTHI, S., AND GAVRILOVSKA, A. Kleio: A hybrid memory page scheduler with machine intelligence. In *Proceedings of the 28th International Symposium on High-Performance Parallel and Distributed Computing* (2019), pp. 37–48.

[41] DUONG, N., ZHAO, D., KIM, T., CAMMAROTA, R., VALERO, M., AND VEIDENBAUM, A. V. Improving cache management policies using dynamic reuse distances. In *Microarchitecture (MICRO), 2012 45th Annual IEEE/ACM International Symposium on* (2012), IEEE, pp. 389–400.

[42] EGAN, C., STEVEN, G., QUICK, P., ANGUERA, R., STEVEN, F., AND VINTAN, L. Two-level branch prediction using neural networks. *Journal of Systems Architecture 49*, 12-15 (2003), 557–570.

[43] ELMOUSTAPHA, O., JAMES, W., CHARLES, Y., AND KSHITIJ, A. On the comparison of regression algorithms for computer architecture performance analysis of software applications. In *Proceedings of the First Workshop on Statistical and Machine learning approaches applied to ARchitectures and compilaTion (SMART'07)* (2007).

[44] EMER, J., AND GLOY, N. A language for describing predictors and its application to automatic synthesis. In *ACM SIGARCH Computer Architecture News* (1997), vol. 25, ACM, pp. 304–314.

[45] FERN, A., GIVAN, R., FALSAFI, B., AND VIJAYKUMAR, T. Dynamic feature selection for hardware prediction. *ECE Technical Reports* (2000), 28.

[46] FETTES, Q., CLARK, M., BUNESCU, R., KARANTH, A., AND LOURI, A. Dynamic voltage and frequency scaling in nocs with supervised and reinforcement learning techniques. *IEEE Transactions on Computers 68*, 3 (2018), 375–389.

[47] FETTES, Q., KARANTH, A., BUNESCU, R., LOURI, A., AND SHIFLETT, K. Hardware-level thread migration to reduce on-chip data movement via reinforcement learning. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems 39*, 11 (2020), 3638–3649.

[48] FOOTS, C. Cpu hardware classification and performance prediction using neural networks and statistical learning. *International Journal of Artificial Intelligence and Applications (IJAIA) 11*, 4 (2020).

[49] FRIEDMAN, J., HASTIE, T., AND TIBSHIRANI, R. *The elements of statistical learning*, vol. 1. Springer series in statistics New York, 2001.

[50] GAO, H., AND ZHOU, H. Adaptive information processing: An effective way to improve perceptron branch predictors. *Journal of Instruction-Level Parallelism 7* (2005), 1–10.

[51] GARZA, E., MIRBAGHER-AJORPAZ, S., KHAN, T. A., AND JIMÉNEZ, D. A. Bit-level perceptron prediction for indirect

branches. In *2019 ACM/IEEE 46th Annual International Symposium on Computer Architecture (ISCA)* (2019), IEEE, pp. 27–38.

[52] GAUDET, C. Using binary neural networks for hardware branch prediction.

[53] GOMATHEESHWARI, B., AND SELVAKUMAR, J. Appropriate allocation of workloads on performance asymmetric multicore architectures via deep learning algorithms. *Microprocessors and Microsystems 73* (2020), 102996.

[54] GOPE, D., AND LIPASTI, M. H. Bias-free neural predictor. *The 4th Championship Branch Prediction, http://www. jilp. org/cbp2014* (2014).

[55] GREENHALGH, P. Big. little processing with arm cortex-a15 & cortex-a7. *ARM White paper 17* (2011).

[56] HELMY, T., AL-AZANI, S., AND BIN-OBAIDELLAH, O. A machine learning-based approach to estimate the cpu-burst time for processes in the computational grids. In *Artificial Intelligence, Modelling and Simulation (AIMS), 2015 3rd International Conference on* (2015), IEEE, pp. 3–8.

[57] HINTON, G. E. Learning multiple layers of representation. *Trends in cognitive sciences 11*, 10 (2007), 428–434.

[58] HO, C. Y., AND FONG, A. S. Combining local and global history hashing in perceptron branch prediction. In *Computer and Information Science, 2007. ICIS 2007. 6th IEEE/ACIS International Conference on* (2007), IEEE, pp. 54–59.

[59] HOSTE, K., PHANSALKAR, A., EECKHOUT, L., GEORGES, A., JOHN, L. K., AND DE BOSSCHERE, K. Performance prediction based on inherent program similarity. In *Parallel Architectures and Compilation Techniques (PACT), 2006 International Conference on* (2006), IEEE, pp. 114–122.

[60] İPEK, E., MCKEE, S. A., CARUANA, R., DE SUPINSKI, B. R., AND SCHULZ, M. *Efficiently exploring architectural design spaces via predictive modeling*, vol. 40. ACM, 2006.

[61] IPEK, E., MUTLU, O., MARTÍNEZ, J. F., AND CARUANA, R. Self-optimizing memory controllers: A reinforcement learning approach. In *Computer Architecture, 2008. ISCA'08. 35th International Symposium on* (2008), IEEE, pp. 39–50.

[62] JAIN, A., AND AMARASINGHE, S. Learning automatic schedulers with projective reparameterization.

[63] JALEEL, A., THEOBALD, K. B., STEELY JR, S. C., AND EMER, J. High performance cache replacement using re-reference interval prediction (rrip). In *ACM SIGARCH Computer Architecture News* (2010), vol. 38, ACM, pp. 60–71.

[64] JAMES, G., WITTEN, D., HASTIE, T., AND TIBSHIRANI, R. *An introduction to statistical learning*, vol. 112. Springer, 2013.

[65] JHA, R., KARUVALLY, A., TIWARI, S., AND MOSS, J. E. B. Cache miss rate predictability via neural networks.

[66] JIA, W., SHAW, K. A., AND MARTONOSI, M. Stargazer: Automated regression-based gpu design space exploration. In *Performance Analysis of Systems and Software (ISPASS), 2012 IEEE International Symposium on* (2012), IEEE, pp. 2–13.

[67] JIMÉNEZ, D. Multiperspective perceptron predictor. *Championship Branch Prediction (CBP-5)* (2016).

[68] JIMÉNEZ, D. A. Snip: Scaled neural indirect predictor.

[69] JIMÉNEZ, D. A. Fast path-based neural branch prediction. In *Proceedings of the 36th annual IEEE/ACM International Symposium on Microarchitecture* (2003), IEEE Computer Society, p. 243.

[70] JIMÉNEZ, D. A. Improved latency and accuracy for neural branch prediction. *ACM Transactions on Computer Systems (TOCS) 23*, 2 (2005), 197–218.

[71] JIMÉNEZ, D. A. Piecewise linear branch prediction. In *ACM SIGARCH Computer Architecture News* (2005), vol. 33, IEEE Computer Society, pp. 382–393.

[72] JIMÉNEZ, D. A. Generalizing neural branch prediction. *ACM Transactions on Architecture and Code Optimization (TACO) 5*, 4 (2009), 17.

[73] JIMÉNEZ, D. A. An optimized scaled neural branch predictor. In *Computer Design (ICCD), 2011 IEEE 29th International Conference on* (2011), IEEE, pp. 113–118.

[74] JIMÉNEZ, D. A. Insertion and promotion for tree-based pseudolru last-level caches. In *Proceedings of the 46th Annual IEEE/ACM International Symposium on Microarchitecture* (2013), ACM, pp. 284–296.

[75] JIMÉNEZ, D. A., AND LIN, C. Perceptron learning for predicting the behavior of conditional branches. In *Neural Networks, 2001. Proceedings. IJCNN'01. International Joint Conference on* (2001), vol. 3, IEEE, pp. 2122–2127.

[76] JIMÉNEZ, D. A., AND LIN, C. Neural methods for dynamic branch prediction. *ACM Transactions on Computer Systems (TOCS) 20*, 4 (2002), 369–397.

[77] JOLLIFFE, I. T. Principal components in regression analysis. *Principal component analysis* (2002), 167–198.

[78] JUAN, D.-C., AND MARCULESCU, D. Power-aware performance increase via core/uncore reinforcement control for chip-multiprocessors. In *Proceedings of the 2012 ACM/IEEE international symposium on Low power electronics and design* (2012), ACM, pp. 97–102.

[79] JUNDT, A., CAUBLE-CHANTRENNE, A., TIWARI, A., PERAZA, J., LAURENZANO, M. A., AND CARRINGTON, L. Compute bottlenecks on the new 64-bit arm. In *Proceedings of the 3rd International Workshop on Energy Efficient Supercomputing* (2015), ACM, p. 6.

[80] JUNG, H., AND PEDRAM, M. Supervised learning based power management for multicore processors. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems 29*, 9 (2010), 1395–1408.

[81] KAELBLING, L. P., LITTMAN, M. L., AND MOORE, A. W. Reinforcement learning: A survey. *Journal of artificial intelligence research 4* (1996), 237–285.

[82] Kelley, S. Branch and confidence prediction using perceptrons.

[83] Kevin, W., Jafanza, V., Kansal, A., Taur, A., Zahran, M., and Muzahid, A. The case for learning application behavior to improve hardware energy efficiency. *arXiv preprint arXiv:2004.13074* (2020).

[84] Khakhaeng, S., and Chantrapornchai, C. On the finding proper cache prediction model using neural network. In *Knowledge and Smart Technology (KST), 2016 8th International Conference on* (2016), IEEE, pp. 146–151.

[85] Khan, S., Xekalakis, P., Cavazos, J., and Cintra, M. Using predictivemodeling for cross-program design space exploration in multicore systems. In *Proceedings of the 16th International Conference on Parallel Architecture and Compilation Techniques* (2007), IEEE Computer Society, pp. 327–338.

[86] Khasawneh, K. N., Abu-Ghazaleh, N., Ponomarev, D., and Yu, L. Rhmd: evasion-resilient hardware malware detectors. In *Proceedings of the 50th Annual IEEE/ACM International Symposium on Microarchitecture* (2017), ACM, pp. 315–327.

[87] Khasawneh, K. N., Ozsoy, M., Donovick, C., Abu-Ghazaleh, N., and Ponomarev, D. Ensemble learning for low-level hardware-supported malware detection. In *International Workshop on Recent Advances in Intrusion Detection* (2015), Springer, pp. 3–25.

[88] Khasawneh, K. N., Ozsoy, M., Donovick, C., Ghazaleh, N. A., and Ponomarev, D. V. Ensemblehmd: Accurate hardware malware detectors with specialized ensemble classifiers. *IEEE Transactions on Dependable and Secure Computing* (2018).

[89] Kim, H.-C., Pang, S., Je, H.-M., Kim, D., and Bang, S. Y. Constructing support vector machine ensemble. *Pattern recognition 36*, 12 (2003), 2757–2767.

[90] Kim, S. Branch prediction using advanced neural methods. Tech. rep., Technical Report, University of California, Berkeley, 2003.

[91] Kim, Y. G., and Wu, C.-J. Autoscale: Energy efficiency optimization for stochastic edge inference using reinforcement learning. In *2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)* (2020), IEEE, pp. 1082–1096.

[92] Kirby, O., Mirabbasi, S., and Aamodt, T. M. Mixed-signal neural network branch prediction, 2007.

[93] Klaine, P. V., Imran, M. A., Onireti, O., and Souza, R. D. A survey of machine learning techniques applied to self-organizing cellular networks. *IEEE Communications Surveys & Tutorials 19*, 4 (2017), 2392–2431.

[94] Kohonen, T. Learning vector quantization. In *Self-Organizing Maps*. Springer, 1995, pp. 175–189.

[95] Leather, H., Bonilla, E., and O'Boyle, M. Automatic feature generation for machine learning based optimizing compilation. In *Code Generation and Optimization, 2009. CGO 2009. International Symposium on* (2009), IEEE, pp. 81–91.

[96] Lee, B. C., and Brooks, D. M. Illustrative design space studies with microarchitectural regression models. In *High Performance Computer Architecture, 2007. HPCA 2007. IEEE 13th International Symposium on* (2007), IEEE, pp. 340–351.

[97] Lee, B. C., Collins, J., Wang, H., and Brooks, D. Cpr: Composable performance regression for scalable multiprocessor models. In *Proceedings of the 41st annual IEEE/ACM International Symposium on Microarchitecture* (2008), IEEE Computer Society, pp. 270–281.

[98] Lee, J.-B. The multiple branch predictor using perceptrons. *The Transactions of The Korean Institute of Electrical Engineers 58*, 3 (2009), 621–626.

[99] Lee, W., Kim, Y., Ryoo, J. H., Sunwoo, D., Gerstlauer, A., and John, L. K. Powertrain: A learning-based calibration of mcpat power models. In *Low Power Electronics and Design (ISLPED), 2015 IEEE/ACM International Symposium on* (2015), IEEE, pp. 189–194.

[100] Leiserson, C. E., Thompson, N. C., Emer, J. S., Kuszmaul, B. C., Lampson, B. W., Sanchez, D., and Schardl, T. B. There's plenty of room at the top: What will drive computer performance after moore's law? *Science 368*, 6495 (2020).

[101] Li, D., Yao, S., and Wang, Y. Processor design space exploration via statistical sampling and semi-supervised ensemble learning. *IEEE Access 6* (2018), 25495–25505.

[102] Li, J., Ma, X., Singh, K., Schulz, M., de Supinski, B. R., and McKee, S. A. Machine learning based online performance prediction for runtime parallelization and task scheduling. In *Performance Analysis of Systems and Software, 2009. ISPASS 2009. IEEE International Symposium on* (2009), IEEE, pp. 89–100.

[103] Li, S., Ahn, J. H., Strong, R. D., Brockman, J. B., Tullsen, D. M., and Jouppi, N. P. 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* (2009), pp. 469–480.

[104] Li, Y., Penney, D., Ramamurthy, A., and Chen, L. Characterizing on-chip traffic patterns in general-purpose gpus: A deep learning approach. In *2019 IEEE 37th International Conference on Computer Design (ICCD)* (2019), IEEE, pp. 56–64.

[105] Liao, S.-w., Hung, T.-H., Nguyen, D., Chou, C., Tu, C., and Zhou, H. Machine learning-based prefetch optimization for data center applications. In *Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis* (2009), ACM, p. 56.

[106] Lin, T.-R., Li, Y., Pedram, M., and Chen, L. Design space exploration of memory controller placement in throughput processors with deep learning. *IEEE Computer Architecture Letters 18*, 1 (2019), 51–54.

[107] Lin, Y., Hafdi, D., Wang, K., Liu, Z., and Han, S. Neural-hardware architecture search.

[108] Lippmann, R. An introduction to computing with neural nets. *IEEE Assp magazine 4*, 2 (1987), 4–22.

[109] Lopes, A. S. B., and Pereira, M. M. A machine learning approach to accelerating dse of reconfigurable accelerator systems. In *2020 33rd Symposium on Integrated Circuits and Systems Design (SBCCI)* (2020), IEEE, pp. 1–6.

[110] Lowe-Power, J., Ahmad, A. M., Akram, A., Alian, M., Amslinger, R., Andreozzi, M., Armejach, A., Asmussen, N., Bharadwaj, S., Black, G., et al. The gem5 simulator: Version 20.0+. *arXiv preprint arXiv:2007.03152* (2020).

[111] Ma, X., Huang, P., Jin, X., Wang, P., Park, S., Shen, D., Zhou, Y., Saul, L. K., and Voelker, G. M. edoctor: Automatically diagnosing abnormal battery drain issues on smartphones. In *NSDI* (2013), vol. 13, pp. 57–70.

[112] Malik, A. M., Russell, T., Chase, M., and Van Beek, P. Learning heuristics for basic block instruction scheduling. *Journal of Heuristics 14*, 6 (2008), 549–569.

[113] Mao, Y., Shen, J., and Gui, X. A study on deep belief net for branch prediction. *IEEE Access* (2017).

[114] Margaritov, A., Ustiugov, D., Bugnion, E., and Grot, B. Virtual address translation via learned page table indexes. In *Conference on Neural Information Processing Systems* (2018).

[115] Mariani, G., Anghel, A., Jongerius, R., and Dittmann, G. Predicting cloud performance for hpc applications: a user-oriented approach. In *Proceedings of the 17th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing* (2017), IEEE Press, pp. 524–533.

[116] McGovern, A., and Moss, J. E. B. Scheduling straight-line code using reinforcement learning and rollouts. In *Advances in neural information processing Systems* (1999), pp. 903–909.

[117] Mendis, C., Renda, A., Amarasinghe, S., and Carbin, M. Ithemal: Accurate, portable and fast basic block throughput estimation using deep neural networks. In *International Conference on Machine Learning* (2019), PMLR, pp. 4505–4515.

[118] Mikolov, T., Sutskever, I., Chen, K., Corrado, G. S., and Dean, J. Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems* (2013), pp. 3111–3119.

[119] Mirbagher-Ajorpaz, S., Pokam, G., Mohammadian-Koruyeh, E., Garza, E., Abu-Ghazaleh, N., and Jiménez, D. A. Perspectron: Detecting invariant footprints of microarchitectural attacks with perceptron. In *2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)* (2020), IEEE, pp. 1124–1137.

[120] Mishra, N., Lafferty, J. D., and Hoffmann, H. Esp: A machine learning approach to predicting application interference. In *Autonomic Computing (ICAC), 2017 IEEE International Conference on* (2017), IEEE, pp. 125–134.

[121] Mittal, S. A survey of techniques for dynamic branch prediction. *arXiv preprint arXiv:1804.00261* (2018).

[122] Moeng, M., and Melhem, R. Applying statistical machine learning to multicore voltage & frequency scaling. In *Proceedings of the 7th ACM international conference on Computing frontiers* (2010), ACM, pp. 277–286.

[123] Monchiero, M., and Palermo, G. The combined perceptron branch predictor. In *European Conference on Parallel Processing* (2005), Springer, pp. 487–496.

[124] Monsifrot, A., Bodin, F., and Quiniou, R. A machine learning approach to automatic production of compiler heuristics. In *AIMSA* (2002), vol. 2, Springer, pp. 41–50.

[125] Mukundan, J., and Martinez, J. F. Morse: Multi-objective reconfigurable self-optimizing memory scheduler. In *High Performance Computer Architecture (HPCA), 2012 IEEE 18th International Symposium on* (2012), IEEE, pp. 1–12.

[126] Murphy, K. P. Naive bayes classifiers. *University of British Columbia 18* (2006).

[127] Murphy, R. C., Wheeler, K. B., Barrett, B. W., and Ang, J. A. Introducing the graph 500. *Cray Users Group (CUG) 19* (2010), 45–74.

[128] Mushtaq, M., Akram, A., Bhatti, M. K., Chaudhry, M., Lapotre, V., and Gogniat, G. Nights-watch: A cache-based side-channel intrusion detector using hardware performance counters. In *Proceedings of the 7th International Workshop on Hardware and Architectural Support for Security and Privacy* (2018), pp. 1–8.

[129] Mushtaq, M., Bricq, J., Bhatti, M. K., Akram, A., Lapotre, V., Gogniat, G., and Benoit, P. Whisper: A tool for run-time detection of side-channel attacks. *IEEE Access 8* (2020), 83871–83900.

[130] Navarro, O., Mori, J., Hoffmann, J., Stuckmann, F., and Hübner, M. A machine learning methodology for cache recommendation. In *International Symposium on Applied Reconfigurable Computing* (2017), Springer, pp. 311–322.

[131] Nemirovsky, D., Arkose, T., Markovic, N., Nemirovsky, M., Unsal, O., and Cristal, A. A machine learning approach for performance prediction and scheduling on heterogeneous cpus. In *Computer Architecture and High Performance Computing (SBAC-PAD), 2017 29th International Symposium on* (2017), IEEE, pp. 121–128.

[132] Ninomiya, Y., and Abe, K. Path traced perceptron branch predictor using local history for weight selection. In *The 2nd JILP Championship Branch Prediction Competition (CBP-2) in conjunction with The 39th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO-39)* (2006), Citeseer.

[133] O'neal, K., Brisk, P., Abousamra, A., Waters, Z., and Shriver, E. Gpu performance estimation using software rasterization and machine learning. *ACM Transactions on Embedded Computing Systems (TECS) 16*, 5s (2017), 148.

[134] O'Neal, K., Liu, M., Tang, H., Kalantar, A., DeRenard, K., and Brisk, P. Hlspredict: Cross platform performance prediction for fpga high-level synthesis. In *2018 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)* (2018), IEEE, pp. 1–8.

[135] Ozsoy, M., Khasawneh, K. N., Donovick, C., Gorelik, I., Abu-Ghazaleh, N., and Ponomarev, D. Hardware-based malware detection using low-level architectural features. *IEEE Transactions on Computers 65*, 11 (2016), 3332–3344.

[136] O'Neal, K., Brisk, P., Shriver, E., and Kishinevsky, M. Hardware-assisted cross-generation prediction of gpus under design. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems 38*, 6 (2018), 1133–1146.

[137] Patel, N., Sasan, A., and Homayoun, H. Analyzing hardware based malware detectors. In *Proceedings of the 54th Annual Design Automation Conference 2017* (2017), ACM, p. 25.

[138] Peled, L., Mannor, S., Weiser, U., and Etsion, Y. Semantic locality and context-based prefetching using reinforcement learning. In *Computer Architecture (ISCA), 2015 ACM/IEEE 42nd Annual International Symposium on* (2015), IEEE, pp. 285–297.

[139] Peled, L., Weiser, U., and Etsion, Y. Towards memory prefetching with neural networks: Challenges and insights. *arXiv preprint arXiv:1804.00478* (2018).

[140] Peled, L., Weiser, U., and Etsion, Y. A neural network prefetcher for arbitrary memory access patterns. *ACM Transactions on Architecture and Code Optimization (TACO) 16*, 4 (2019), 1–27.

[141] Pelikan, M., Goldberg, D. E., and Cantú-Paz, E. Boa: The bayesian optimization algorithm. In *Proceedings of the 1st Annual Conference on Genetic and Evolutionary Computation-Volume 1* (1999), Morgan Kaufmann Publishers Inc., pp. 525–532.

[142] Pricopi, M., Muthukaruppan, T. S., Venkataramani, V., Mitra, T., and Vishin, S. Power-performance modeling on asymmetric multi-cores. In *Compilers, Architecture and Synthesis for Embedded Systems (CASES), 2013 International Conference on* (2013), IEEE, pp. 1–10.

[143] Prodromou, A., Venkat, A., and Tullsen, D. M. Deciphering predictive schedulers for heterogeneous-isa multicore architectures. In *Proceedings of the 10th International Workshop on Programming Models and Applications for Multicores and Manycores* (2019), pp. 51–60.

[144] Rahman, S., Burtscher, M., Zong, Z., and Qasem, A. Maximizing hardware prefetch effectiveness with machine learning. In *High Performance Computing and Communications (HPCC), 2015 IEEE 7th International Symposium on Cyberspace Safety and Security (CSS), 2015 IEEE 12th International Conferen on Embedded Software and Systems (ICESS), 2015 IEEE 17th International Conference on* (2015), IEEE, pp. 383–389.

[145] Reagen, B., Hernández-Lobato, J. M., Adolf, R., Gelbart, M., Whatmough, P., Wei, G.-Y., and Brooks, D. A case for efficient accelerator design space exploration via bayesian optimization. In *Low Power Electronics and Design (ISLPED, 2017 IEEE/ACM International Symposium on* (2017), IEEE, pp. 1–6.

[146] Reddy, B. K., Walker, M. J., Balsamo, D., Diestelhorst, S., Al-Hashimi, B. M., and Merrett, G. V. Empirical cpu power modelling and estimation in the gem5 simulator. In *Power and Timing Modeling, Optimization and Simulation (PATMOS), 2017 27th International Symposium on* (2017), IEEE, pp. 1–8.

[147] Renda, A., Chen, Y., Mendis, C., and Carbin, M. Difftune: Optimizing cpu simulator parameters with learned differentiable surrogates. In *2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)* (2020), IEEE, pp. 442–455.

[148] Rethinagiri, S. K., Palomar, O., Ben Atitallah, R., Niar, S., Unsal, O., and Kestelman, A. C. System-level power estimation tool for embedded processor based platforms. In *Proceedings of the 6th Workshop on Rapid Simulation and Performance Evaluation: Methods and Tools* (2014), ACM, p. 5.

[149] Ribas, L. V. M., and Goncalves, R. Evaluating branch prediction using two-level perceptron table. In *Parallel, Distributed, and Network-Based Processing, 2006. PDP 2006. 14th Euromicro International Conference on* (2006), IEEE, pp. 4–pp.

[150] Rodrigues, R., Annamalai, A., Koren, I., and Kundu, S. A study on the use of performance counters to estimate power in microprocessors. *IEEE Transactions on Circuits and Systems II: Express Briefs 60*, 12 (2013), 882–886.

[151] Saadeldeen, H., Franklin, D., Long, G., Hill, C., Browne, A., Strukov, D., Sherwood, T., and Chong, F. T. Memristors for neural branch prediction: a case study in strict latency and write endurance challenges. In *Proceedings of the ACM International Conference on Computing Frontiers* (2013), ACM, p. 26.

[152] Sanchez, E. O., and Sun, X.-H. Cads: Core-aware dynamic scheduler for multicore memory controllers. *arXiv preprint arXiv:1907.07776* (2019).

[153] Seber, G. A., and Lee, A. J. *Linear regression analysis*, vol. 329. John Wiley & Sons, 2012.

[154] Sen, S., and Imam, N. Machine learning based design space exploration for hybrid main-memory design. In *Proceedings of the International Symposium on Memory Systems* (2019), pp. 480–489.

[155] Seng, J., and Hamerly, G. Exploring perceptron-based register value prediction. In *Second Value Prediction and Value-Based Optimization Workshop, held in conjuction with ASPLOS* (2004).

[156] Sethuram, R., Khan, O. I., Venkatanarayanan, H. V., and Bushnell, M. L. A neural net branch predictor to reduce power. In *VLSI Design, 2007. Held jointly with 6th International Conference on Embedded Systems., 20th International Conference on* (2007), IEEE, pp. 679–684.

[157] Seznec, A. Revisiting the perceptron predictor. *PI-1620, IRISA, May* (2004).

[158] Seznec, A. Tage-sc-l branch predictors. In *JILP-Championship Branch Prediction* (2014).

[159] Sherwood, T., Perelman, E., Hamerly, G., and Calder, B. Automatically characterizing large scale program behavior. In *ACM SIGARCH Computer Architecture News* (2002), vol. 30, ACM, pp. 45–57.

[160] Shi, G., and Lipasti, M. Perceptron branch prediction with separated taken/not-taken weight tables.

[161] Shi, Z., Huang, X., Jain, A., and Lin, C. Applying deep learning to the cache replacement problem. In *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture* (2019), pp. 413–425.

[162] Shi, Z., Jain, A., Swersky, K., Hashemi, M., Ranganathan, P., and Lin, C. A neural hierarchical sequence model for irregular data prefetching.

[163] SPEC. SPEC CPU 2006, 2006. Retrieved August 5, 2015 from https://www.spec.org/cpu2006/.

[164] Srinivas, M., and Patnaik, L. M. Genetic algorithms: A survey. *computer 27*, 6 (1994), 17–26.

[165] Srinivasan, R., Frachtenberg, E., Lubeck, O., Pakin, S., and Cook, J. An idealistic neuro-ppm branch predictor. *Journal of Instruction-Level Parallelism 9* (2007), 1–13.

[166] St Amant, R., Jiménez, D. A., and Burger, D. Low-power, high-performance analog neural branch prediction. In *Proceedings of the 41st annual IEEE/ACM International Symposium on Microarchitecture* (2008), IEEE Computer Society, pp. 447–458.

[167] Stanley, T. J., and Mudge, T. N. A parallel genetic algorithm for multiobjective microprocessor design. In *ICGA* (1995), pp. 597–604.

[168] Strickland, J. *Data Analytics using Open-Source Tools*. Lulu. com, 2015.

[169] Sutton, R. S., and Barto, A. G. *Reinforcement learning: An introduction*, vol. 1. MIT press Cambridge, 1998.

[170] Tarjan, D., and Skadron, K. Merging path and gshare indexing in perceptron branch prediction. *ACM transactions on architecture and code optimization (TACO) 2*, 3 (2005), 280–300.

[171] Tarsa, S. J., Chowdhury, R. B. R., Sebot, J., Chinya, G., Gaur, J., Sankaranarayanan, K., Lin, C.-K., Chappell, R., Singhal, R., and Wang, H. Post-silicon cpu adaptation made practical using machine learning. In *2019 ACM/IEEE 46th Annual International Symposium on Computer Architecture (ISCA)* (2019), IEEE, pp. 14–26.

[172] Tarsa, S. J., Lin, C.-K., Keskin, G., Chinya, G., and Wang, H. Improving branch prediction by modeling global history with convolutional neural networks. *arXiv preprint arXiv:1906.09889* (2019).

[173] Teran, E., Wang, Z., and Jiménez, D. A. Perceptron learning for reuse prediction. In *Microarchitecture (MICRO), 2016 49th Annual IEEE/ACM International Symposium on* (2016), IEEE, pp. 1–12.

[174] Tu, J., Chen, J., and John, L. K. Hardware efficient piecewise linear branch predictor. In *VLSI Design, 2007. Held jointly with 6th International Conference on Embedded Systems., 20th International Conference on* (2007), IEEE, pp. 673–678.

[175] Vapnik, V. *Statistical learning theory. 1998.* Wiley, New York, 1998.

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

[177] Verma, S. Perceptron learning driven coherence aware reuse prediction for last-level caches.

[178] Vietri, G., Rodriguez, L. V., Martinez, W. A., Lyons, S., Liu, J., Rangaswami, R., Zhao, M., and Narasimhan, G. Driving cache replacement with ml-based lecar. In *10th {USENIX} Workshop on Hot Topics in Storage and File Systems (HotStorage 18)* (2018).

[179] Vintan, L. N., and Iridon, M. Towards a high performance neural branch predictor. In *Neural Networks, 1999. IJCNN'99. International Joint Conference on* (1999), vol. 2, IEEE, pp. 868–873.

[180] Walker, M. J., Das, A. K., Merrett, G. V., and Hashimi, B. Run-time power estimation for mobile and embedded asymmetric multi-core cpus.

[181] Walker, M. J., Diestelhorst, S., Hansson, A., Das, A. K., Yang, S., Al-Hashimi, B. M., and Merrett, G. V. Accurate and stable run-time power modeling for mobile and embedded cpus. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems 36*, 1 (2017), 106–119.

[182] Wang, H., and Luo, Z. Data cache prefetching with perceptron learning. *arXiv preprint arXiv:1712.00905* (2017).

[183] Wang, K., and Louri, A. Cure: A high-performance, low-power, and reliable network-on-chip design using rein-forcement learning. *IEEE Transactions on Parallel and Distributed Systems 31*, 9 (2020), 2125–2138.

[184] Wang, S., and Ipek, E. Reducing data movement energy via online data clustering and encoding. In *Microarchitecture (MICRO), 2016 49th Annual IEEE/ACM International Symposium on* (2016), IEEE, pp. 1–13.

[185] Wang, Y., and Chen, L. Dynamic branch prediction using machine learning. *ECS-201A, Fall* (2005).

[186] Wang, Z., and O'Boyle, M. F. Mapping parallelism to multi-cores: a machine learning based approach. In *ACM Sigplan notices* (2009), vol. 44, ACM, pp. 75–84.

[187] Wang, Z., Tournavitis, G., Franke, B., and O'boyle, M. F. Integrating profile-driven parallelism detection and machine-learning-based mapping. *ACM Transactions on Architecture and Code Optimization (TACO) 11*, 1 (2014), 2.

[188] Won, J.-Y., Chen, X., Gratz, P., Hu, J., and Soteriou, V. Up by their bootstraps: Online learning in artificial neural networks for cmp uncore power management. In *High Performance Computer Architecture (HPCA), 2014 IEEE 20th*

International Symposium on (2014), IEEE, pp. 308–319.

[189] Wu, G., Greathouse, J. L., Lyashevsky, A., Jayasena, N., and Chiou, D. Gpgpu performance and power estimation using machine learning. In *High Performance Computer Architecture (HPCA), 2015 IEEE 21st International Symposium on* (2015), IEEE, pp. 564–576.

[190] Xi, S. L., Jacobson, H., Bose, P., Wei, G.-Y., and Brooks, D. Quantifying sources of error in mcpat and potential impacts on architectural studies. In *High Performance Computer Architecture (HPCA), 2015 IEEE 21st International Symposium on* (2015), IEEE, pp. 577–589.

[191] Yang, S., Shafik, R. A., Merrett, G. V., Stott, E., Levine, J. M., Davis, J., and Al-Hashimi, B. M. Adaptive energy minimization of embedded heterogeneous systems using regression-based learning. In *Power and Timing Modeling, Optimization and Simulation (PATMOS), 2015 25th International Workshop on* (2015), IEEE, pp. 103–110.

[192] Yarom, Y., and Falkner, K. Flush+ reload: A high resolution, low noise, l3 cache side-channel attack. In *USENIX Security Symposium* (2014), pp. 719–732.

[193] Yazdanbakhsh, A., Park, J., Sharma, H., Lotfi-Kamran, P., and Esmaeilzadeh, H. Neural acceleration for gpu throughput processors. In *Microarchitecture (MICRO), 2015 48th Annual IEEE/ACM International Symposium on* (2015), IEEE, pp. 482–493.

[194] Ye, R., and Xu, Q. Learning-based power management for multicore processors via idle period manipulation. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems 33*, 7 (2014), 1043–1055.

[195] Yin, J., Sethumurugan, S., Eckert, Y., Patel, C., Smith, A., Morton, E., Oskin, M., Jerger, N. E., and Loh, G. H. Experiences with ml-driven design: A noc case study. In *2020 IEEE International Symposium on High Performance Computer Architecture (HPCA)* (2020), IEEE, pp. 637–648.

[196] Zhang, L., Wu, N., Ge, F., Zhou, F., and Yahya, M. R. A dynamic branch predictor based on parallel structure of srnn. *IEEE Access 8* (2020), 86230–86237.

[197] Zheng, H., and Louri, A. An energy-efficient network-on-chip design using reinforcement learning. In *Proceedings of the 56th Annual Design Automation Conference 2019* (2019), pp. 1–6.

[198] Zheng, X., John, L. K., and Gerstlauer, A. Accurate phase-level cross-platform power and performance estimation. In *Proceedings of the 53rd Annual Design Automation Conference* (2016), ACM, p. 4.

[199] Zheng, X., Ravikumar, P., John, L. K., and Gerstlauer, A. Learning-based analytical cross-platform performance prediction. In *Embedded Computer Systems: Architectures, Modeling, and Simulation (SAMOS), 2015 International Conference on* (2015), IEEE, pp. 52–59.

[200] Zhu, L., Chen, C., Su, Z., Chen, W., Li, T., and Yu, Z. Bbs: Micro-architecture benchmarking blockchain systems through machine learning and fuzzy set. In *2020 IEEE International Symposium on High Performance Computer Architecture (HPCA)* (2020), IEEE, pp. 411–423.

A AN OVERVIEW OF ML TECHNIQUES

Machine Learning (ML) refers to the process in which computers learn to make decisions based on the given data set without being explicitly programmed to do so [8]. There are numerous ML algorithms that can be grouped into three categories: supervised learning algorithms, unsupervised learning algorithms and other types of algorithms. Learning algorithms are usually trained on a set of samples called a "training set". A different set of data is usually used to test the accuracy of decisions made by the learning algorithm known as "test set".

This section briefly discusses different ML techniques that are used by the research surveyed in this paper. Readers who wish to have detailed explanation of the discussed methodologies in this section can refer to [64]. A common textbook classification of ML algorithms is shown in Figure 3.

A.1 Supervised Learning

The process of training machine learning models given a set of samples of input vectors with the corresponding output/target vectors is called Supervised learning [32]. The trained model can then be used to predict ouput vectors given a new set of input vectors. The output/target vectors are also known as labels and input data as labelled data.

Machine learning models can operate on different types of data. If the target labels in a data set can be a set of discrete classes/categories, the modeling task is referred as classification task. On the other hand if the target labels in a data set are continous variables, the modeling task is called a
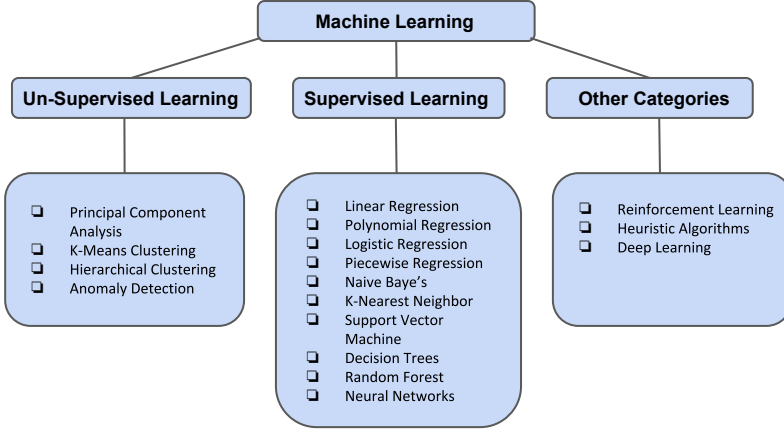
Fig. 3. Classification of common ML algorithms

regression task [64]. In other words, in classfication the built models predict qualitative response and in regression the built models predict quantitative response.

Many of the ML algorithms can be applied to both regression and classification tasks, while there are others that can only be used for classification or regression tasks.

A.1.1 Linear Regression. Linear regression is used to relate different variables and approximates the effect of changing a variable on other variables. Assume that there exists a predictor variable X (also known as independent/input variable) and a corresponding quantitative response Y (also known as dependent/output variable). Linear regression, assuming that there exists a linear relation between dependent and independent variables, generates a model as shown in (1) [64, 153].

$$Y = \alpha_0 + \alpha_1 X \tag{1}$$

Here α_0 and α_1 are regression coefficients unknown before training the model using given input data.

A.1.2 Multiple Linear Regression. In case the ouput variable Y is dependent on more than one independent variables or predictors, the linear regression equation can be extended to include all those predictors. Assuming there are p independent variables the multiple linear regression model would be given as in (2) [64].

$$Y = \alpha_0 + \alpha_1 X_1 + \alpha_2 X_2 + \alpha_3 X_3 + \alpha_4 X_4 + ... + \alpha_m X_m \tag{2}$$

Here α_0 to α_m are regression coefficients unknown before training the model using given input/output data.

A.1.3 Polynomial Regression. If there exists a non-linear relation between response and predictor variables, polynomials of predictor variables can be added in the regression model resulting into Polynomial Regression. For example if a quadratic relation exists between input X and output Y the regression model would become as in (3) [64].

$$Y = \alpha_0 + \alpha_1 X + \alpha_2 X^2 \tag{3}$$

A.1.4 Logistic Regression. Logistic regression algorithm finds its use purely for classification problems. In case of logistic regression the built model predicts the probability that the response Y would be member of particular class. Given an input/independent variable X, logistic regression model can be represented by the logistic function of (4) [64].

$$p(X) = e^{\alpha_0 + \alpha_1 X}/(1 + e^{\alpha_0 + \alpha_1 X}) \tag{4}$$

This relation can also be extended to multiple logistic regression in case of more than one independent variables as in the case of multiple linear regression.

A.1.5 Piecewise Polynomial Regression. Piecewise polynomial regression tries to avoid use of high degree polynomials. Instead it uses models built by low-degree polynomials, each corresponding to a different region of the variable X. As an example, "piecewise cubic polynomial" will use cubic regression models shown in (5) and (6) for different ranges of x. The points of changes in values of coefficients are known as knots. The given example in (5) and (6) has a single knot at point c.

$$y_i = \alpha_{01} + \alpha_{11}x_i + \alpha_{21}x_i{}^2 + \alpha_{31}x_i{}^3 + \epsilon_i \qquad if \ \ x_i < c; \tag{5}$$
$$y_i = \alpha_{02} + \alpha_{12}x_i + \alpha_{22}x_i{}^2 + \alpha_{32}x_i{}^3 + \epsilon_i \qquad if \ \ x_i >= c; \tag{6}$$

A.1.6 Naive Bayes'. Bayes' theorem is used significantly to compute conditional probability. Mathematical form of bayes' theorem is given in (7) [175].

$$Pr(h|e) = Pr(e|h)Pr(h)/Pr(e) \tag{7}$$

Here Pr(h|e) represents the probability that the hypothesis h will be true if some event e exists (posterior probability). Pr(e|h) represents the probability that the event e would occur in presence of an hypothesis h. Pr(h) is the proability of hypothesis by itself without the event (prior probability) and Pr(e) is the probability of the event e [175]. Bayes' theorem is largely used for classification purposes in ML. Bayes' theorem based classifiers that assume that the given inputs/features are independent of each other are known as *Naive Bayes' Classifiers*. Usually, a small number of data points are sufficient to train Naive Bayes' classifiers. They are famous for their scalability and efficiency. A more detailed review of Bayes' classifiers can be found in [126].

A.1.7 K-Nearest Neighbors. K-Nearest Neighbors (KNN) is used to classify unseen data points by observing K already-classified data points that are closest to the unseen data point. Different distance measures can caclculate the distance of new/unseen data point from neighboring data points such as Euclidean, Manhattan, Hamming etc. A mathematical representation of KNN is given in (8). Assuming that there is a test observation that needs to be classified, and K closest points are represented by N_0, then to find out if test point belongs to class j KNN relies on (8). In (8), a fraction of points in N_0 belonging to class j gives the conditional probability of class j.

$$1/K \sum_{i \epsilon N_0} I(y_i = j) \tag{8}$$

Figure 4 shows an example of how KNN algorithm works. For K=8, the data point in question will be classified as 'X' as most of its neighbors belong to category 'X'. KNN is specifically useful for ML problems where "underlying joint distribution of the result and observation" is unknown. KNN does not scale well with larger data sets since it makes use of the complete training data to
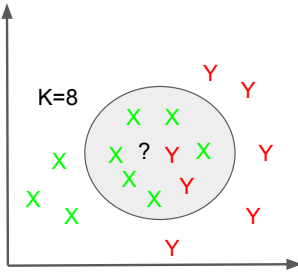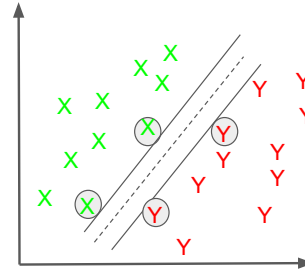
Fig. 4. KNN

Fig. 5. SVM

make new predictions for test data. KNN has also been used for regression problems [32]. More information on KNN can be found in [35].

A.1.8   Support Vector Machines. Support Vector Machine (SVM) is a non-probabilistic ML model that has been used for both classification and regression problems, but mostly for classification. Assuming that we have n number of features or input variables, SVM classifier plots given data points in an n-dimensional space. The algorithm finds hyper-planes in this n-dimensional space which can distinguish given classes with the largest margin. This hyperplane can be a linear/non-linear function of input variables resulting into linear/non-linear SVM. SVM based classifiers use "a few training points" (also known as support vectors) while classifying new data points. Theoretically, support vectors are the most difficult points to classify since they are the nearest to the "decision surface". Maximum margin between these support vectors ensures that the distance between classes is maximized. Figure 5 shows a linear SVM classifier, distinguishing between two classes of data.

SVMs are effective for high number of features and are memory-efficient. However, these models are not very interpretable [89]. More information on SVMs can be found in [21, 64]

A.1.9   Decision Trees. Decision trees summarize the relationship between different values of given features (represented by branches) and the conlusions/results about the output variables's value (represnted by leaves) [2]. Decision trees can be used for both classification and regression problems. Figure 6 shows an example of a decision tree used for a classification problem; is a person ready to run a marathon?

Decision trees are easy to interpret because of their pictorial nature. Generally decision trees are considered to be less accuarate compared to other advanced ML techniques. Interested readers can obtain more information about decision trees in [24, 64].

A.1.10   Random Forests. Random forests form a large number of decision trees based on the training data and then uses the combined behavior of those trees to make any predictions [3]. Only a subset of the entire training data is used to train individual trees. The subset of the training data and input features are selected randomly. Random forests can be used for both classification and regression. Random forests can solve the decision trees' problem of overfitting as they use an ensemble of decision trees.

Random forests are known for their higher accuracy, however they are not very interpretable [28]. More information on random forests can be found in [23, 64].

A.1.11   Neural Networks. Neural networks try to mimic the behavior of human brains. Similar to neurons in brain, neural networks have nodes that perform basic computations using activation functions. Mostly used activation functions are binary and sigmoid ones. Nodes of the neural
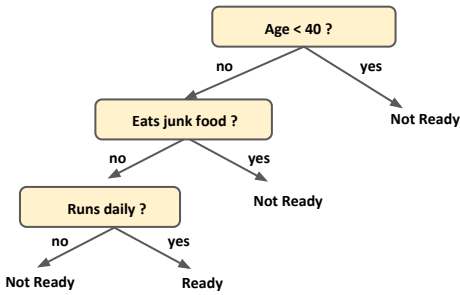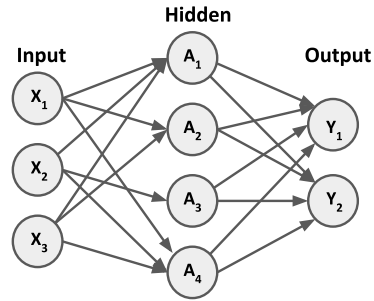
Fig. 6. Decision Tree

Fig. 7. Neural Network

network connect to each other using weighted edges. Neural networks are composed of three different layers: input layer (neurons in this layer get input from outer world), output layer (neurons in this layer generate predicted output response) and hidden layer(this layer performs useful transformations on input data and feed it to the output layer). There can be any number of hidden layers in a neural network. Figure 7 shows an example of a basic neural network, with only one hidden layer. This type of neural networks in which connections between layers move in forward direction only are known as feed-forward neural networks. Neural networks can have variable number of nodes and hidden layers.

Different architectures of neural networks like perceptron, multilayer perceptron, recurrent neural network and boltzman machine network have been proposed. Neural networks learn by configuring the weights of edges and thresholds of activation functions iteratively to achieve the required results. There are two main learning algorithms to train neural networks: *Gradient Descent:* In this learning algorithm the weights of the NN are modified to minimize the difference between the actual outputs and the predicted outputs. *Back propagation:* In this learning algorithm the dissimilarity of the predicted and actual outputs is computed at the output layer and transferred to the input layer using hidden layer.

Neural networks find their use in both regression and classification problems. Neural networks are known for their good accuracy. More information on neural networks can be found in [49, 108]

A.2 Un-Supervised Learning

The process of training machine learning models given only sets of input vectors and no output/target vectors is called Un-supervised learning [32]. In other words, the training data is not labelled in case of Unsupervised learning. This type of learning is used to attain a better understanding of the given data by identifying existing patterns in the data.

A.2.1 Prinicpal Component Analysis. Principal component analysis (PCA) is largely used to reduce the number of dimensions (dimensionality reduction). It allows to understand exisintg patterns in given data by using small number of "representative" variables (called principal components) from a larger number of given "correlated" variables. Basically, PCA transforms the given data to another space such that there is maximum variance among variables in new space. The components which have least variance are discarded as they do not contain much information. More information on PCA can be found in [77].

A.2.2 Clustering. Another popular un-supervised learning technique is clustering i.e. finding groups of data in given unlabelled data-set. Following are two main types of clustering algorithms:

K-Means Clustering: This algorithm starts with specification of required clusters K. Random data points are chosen as centoids of K clusters. The algorithm then identifies the points nearest to the centroid by using some distance measure, calculates mean of all points and assign a new centre to the cluster. The algorithm keeps on identifying closest points and calculate new centres untill a convergence condition is met. More information on K-means clustering can be found in [49, 64].

Heirarchical clustering: Heirarchical clustering does not require the specification of total number of clusters in advance. The algorithm builds a data binary tree that repeatedly combines similar data points. The mostly used form of heirachical clustering known as agglomerative clustering is performed as follows: Each single data point forms its own group. Two closest groups are combined iteratively, untill the time when all data points are contained in a single group/cluster.

More information on heirarchical clustering can be found in [49, 64].

A.2.3   Anomaly Detection. Anamoly detection algorithms is a class of algorithms that identify data points with abnormal behavior in given data set. Many of the anamoly detection algorithms are unsupervised but they can be supervised and semisupervised as well [30].

A.3   Other Types of ML Techniques

A.3.1   Reinforcement Learning. Reinforcement learning is based on an "agent" attached/associated to an "environement" [81]. Agent decides to take certain actions depending on the state of the environment. The changes in the state because of agent's actions are fed back to the agent using reinforcement signal. Depending on the consequences of earlier decision agent receives a *reward* or a *penalty*. Agent should always take actions that try to increase the overall reward [81], which it tries to learn by trial and error using different algorithms.

Q-learning algorithm is one of the largely used Reinforcement Learning (RL) algorithms. Interested readers can read more about reinforcement learning in [169].

A.3.2   Heuristic Algorithms. These type of ML algorithms use rules or heuristics while making decisions. They work well if the solution to a problem is expensive [93]. One of their famous types is Genetic Alorithms. These algorithms take their inspiration from nature. They use a process similar to evolution to find the best working solution to a problem. More information on genetic algorithms can be found in [164].

A.4   Techniques Related to Model Assessment

There are various techniques used to assess a machine learning model. Following is a description of few of them:

A.4.1   Validation Set Approach. This method involves partitioning the given data into two halves: training and validation/test sets. The training data set us used to train the Machine learning model. The trained model is then used for prediction of outputs using the available test set data. The error in test data is usually estimated through MSE (mean squared error). A potential problem with this approach is that the test error can have high variations [64].

A.4.2   Leave-One-Out Cross-Validation. Consider that there are n total observations, then in LOOCV one observation is excluded and the model is trained on the n-1 observations. The excluded observation is used to caluclate test error. All the observations are included in the test set one by one and as a result n test errors are calculated by using training on other observations. LOOCV has less bias and randomness compared to validation set approach [64]. This method can take long time to process if the number of observations is large.

A.4.3 k-fold Cross-Validation. In this apprach the available data is randomly divided into k partitions/groups of equal size. The machine learning model under consideration is trained using k-1 groups of data and then tested on kth group of data by calculating MSE. The aforementioned process is then replicated k times using new/different validation data sets. The overall error is estimated by calculating average of k MSE values. This method is less expensive comapred to LOOCV [64].

A.4.4 Bootstrapping (Bootstrap aggregating). Assuming that there is an n-sized training set called D, bagging is a process of creating m training sets D_i each of size n'. These samples (also called bootstrap samples) are taken from D "uniformly with replacement". This means some data points can be repeated in D_i. If n'=n and n is large, D_i will have nearly 63% of unique samples of D and the others will be duplicates [168]. These m samples are used to train m ML models whose outputs are averaged to combine all of them. Bootstrapping can reduce the variance in results, but results can be more biased [64].

A.5 Techniques Related to Fitting of ML Models

There are various techniques used to train a ML model. Some of them are discussed below:

A.5.1 Subset Selection. This involves using only a subset of the available predictors/inputs/features to fit the ML model [64].

A.5.2 Best subset selection. In this method, every combination of input features is used to train the ML model and the best combination is identified. This method is resource intensive [64].

A.5.3 Step-wise selection. Step-wise selection is more efficient way for subset selection compared to best subset selection. Forward step-wise selection starts with no predictors and keeps on adding predictors one by one untill all predictors are included in the model. At every step, the predictor that leads to greatest improvement in the model is selected. Backward step-wise selection starts with a ML model with all predictors/input varibales and removes the predictors with least benefit one by one [64]. Often hybrid versions of these two models are used.

A.5.4 Shrinkage methods. To reduce variance in a regression model and avoid overfitting, often the model is trained with constraining or regularization of regression cofficient estimates. These constraints move the estimates close to zero. Such methods are known as shrinkage methods.

A.5.5 Ridge Regression. In ridge regression the "coefficient estimates" are the values that minimize the expression in (9) [64].

$$\sum_{i=1}^{N}(y_i - \beta_0 - \sum_{j=1}^{p}\beta_j x_{ij})^2 + \lambda \sum_{j=1}^{p}\beta_j^2 \tag{9}$$

Here λ is a "tuning parameter", p is the number of features and N is the number of data points. Ridge regression (like least sqaures) tries to find coefficients that make "RSS small" which is accounted by first term in (9). The second term in (9) is known as shrinkage penalty which controls the effect of two terms of equation "on the regression cofficient estimates". This is small when β cofficients are close to zero [64].

A.5.6 Lasso Regression. Lasso regression is an alternative to ridge regression. It tries to find coefficients that minimize (10) [64].

$$\sum_{i=1}^{N}(y_i - \beta_0 - \sum_{j=1}^{p} \beta_j x_{ij})^2 + \lambda \sum_{j=1}^{p} |\beta_j| \tag{10}$$

The difference in ridge and lasso regression is in the second term, where lasso regression has β coefficients of ridge regression with absolute signs. Lasso regression not only punishes high values of β like ridge regression, but it makes them zero if they seem irrelevant. This gives lasso regression its variable selection property and makes it more interpretable [64].