A Survey of Machine Learning for Computer Architecture and Systems

NAN WU and YUAN XIE, University of California, Santa Barbara

It has been a long time that computer architecture and systems are optimized to enable efficient execution of machine learning (ML) algorithms or models. Now, it is time to reconsider the relationship between ML and systems, and let ML transform the way that computer architecture and systems are designed. This embraces a twofold meaning: the improvement of designers' productivity, and the completion of the virtuous cycle. In this paper, we present a comprehensive review of work that applies ML for system design, which can be grouped into two major categories, ML-based modelling that involves predictions of performance metrics or some other criteria of interest, and ML-based design methodology that directly leverages ML as the design tool. For ML-based modelling, we discuss existing studies based on their target level of system, ranging from the circuit level to the architecture/system level. For ML-based design methodology, we follow a bottom-up path to review current work, with a scope of (micro-)architecture design (memory, branch prediction, NoC), coordination between architecture/system and workload (resource allocation and management, data center management, and security), compiler, and design automation. We further provide a future vision of opportunities and potential directions, and envision that applying ML for computer architecture and systems would thrive in the community.

CCS Concepts: • **Computing methodologies** → **Machine learning**; • **Computer systems organization** → **Architectures**; • **General and reference** → **Surveys and overviews**.

Additional Key Words and Phrases: machine learning for computer architecture and systems

1 INTRODUCTION

Machine learning (ML) has been doing wonders in many fields, including computer vision [81, 207, 213], speech recognition [76, 83], natural language processing [38, 146, 210], drug discovery [148, 198], robotics [77, 86, 140], playing video games [15, 167, 226], and many other domains [103, 128, 195, 206]. Under some circumstances, ML is capable to reach or surpass human performance. For example, ResNet [81] achieves a better top-5 error rate than that of human on the large scale ImageNet dataset; AlphaGo Zero can beat human professional Go players [206]; there has also made significant progress in training artificial agents playing video games, from single-player games (e.g. Atari [167]) to multi-player games (e.g. StarCraft II [226] and Dota 2 [15]).

Current ML models, most of which are deep neural networks (DNNs) and their variants (e.g. multi-layer perceptrons, convolutional neural networks, and recurrent neural networks), already have high demands of memory and computational resource. As people are seeking better artificial intelligence, there is a trend towards larger, more expressive and more complex models. With diminishing gains brought by the Moore's Law, this trend urges advancements in computer architecture/system for faster and more energy-efficient implementations of ML models. Aiming at ML workloads, improvements have been made in different levels of system and architecture designs. In the algorithm level, pruning, quantization and compression of ML models [79, 92] are applied to eliminate computation complexity and improve hardware efficiency; in the hardware level, there is a renaissance of processing in memory (PIM) and near-data processing (NDP) [12, 73, 179], there also arise specialized architectures and accelerators, ranging from those specifically optimized for convolutional neural networks (CNNs) (e.g. ShiDianNao [57], Eyeriss [31] and SCNN [178]) to those designed for general-purpose DNN acceleration (e.g. DaDianNao [30], TPU [108], and DNPU [204]); in the device level, applying emerging non-volatile memory technologies in architecture

Authors' address: Nan Wu, nanwu@ucsb.edu; Yuan Xie, yuanxie@ucsb.edu, University of California, Santa Barbara, Santa Barbara, California, 93106.

design, such as resistive random-access memory (ReRAM) [234], phase-change memory (PCM) [25], spin-transfer torque magnetic random-access memory (STT-MRAM) [85], which can integrate computation and memory together, provides another promising alternative (e.g. PRIME [35], ISAAC [200] and Resparc [7]).

Driven by increasingly complicated workloads and their diverse performance, accuracy, and power targets, it is non-trivial and laborious to design architecture/systems. Usually these designs are made by human experts based on intuitions and heuristics, which requires expertise in both ML and architecture/system and where great scalability and optimal results cannot be guaranteed especially in the case of more complicated systems. As such, it seems natural to move towards more automated and powerful methodologies for architecture and system designs, and the relationship between ML and system design is being reconsidered. Conventionally, architectural and system optimizations are conducted to accelerate the execution and improve the performance of ML models, and it is undeniable that revolutions in ML to some extent do count on advancements of processing capabilities, e.g. better utilization of parallelism, data reuse and sparsity, etc. Recently, there have been signs of emergence of applying ML to enhance system designs, indicating promising potentials. Applying ML for system designs embraces a twofold meaning: ① the reduction of burdens on human experts designing systems manually so as to improve designers' productivity, and ② the close of the positive feedback loop, i.e., architectures/systems for ML and simultaneously ML for architecture/system, formulating a virtuous cycle to encourage improvements in both sides.

Generally, existing work related to applying ML for system designs falls into two categories. ① ML techniques are employed for **system modelling, which involves performance metrics or some criteria of interest** (e.g. power consumption, latency, throughput, etc.). During the process of designing systems, it is necessary to make fast and accurate predictions of system behaviors. Traditionally, the system modelling is achieved through the forms of cycle-accurate or functional virtual platforms, and instruction set simulators (ISSs) (e.g. gem5 [18], Simics [150]). Even though these methods can provide accurate estimations, they also bring expensive computational costs associated with performance modeling, limiting the scalability to large-scale and complex systems; meanwhile, the long simulation time constrains designers' talents, since only small subsets of the full design space can be explored. ② ML techniques are employed as **a design methodology to directly enhance architecture/system designs**. ML is skilled at extracting features, making decisions without explicit programming, and improving itself automatically with experience. Therefore, applying ML techniques as designs tools has great capabilities to explore design space more proactively and intelligently, manage resource through better understanding of resources' complicated, non-linear interactions, etc., which is possible to deliver true optimal solutions.

In this paper, we present an overview of applying ML for computer architecture/systems, and summarize what system problems can be solved by ML techniques and how ML techniques resolve them, as illustrated in Figure 4. We also discuss challenges and future prospects of applying ML for system designs. This paper is organized as follows. Section 2 is a brief introduction of common ML techniques; Section 3 reviews studies that employ ML techniques for system modelling, from the circuit level to the architecture/system level level; Section 4 presents studies that employ ML techniques as design tools to directly enhance architecture/system designs, with a scope of (micro-)architecture design (memory, branch prediction, NoC), coordination between architecture/system and workload (resource allocation and management, data center management, and security), compiler, and design automation; Section 5 discusses challenges and future prospects of applying ML for system designs, to convey insights of design considerations; Section 6 concludes this paper.

2 DIFFERENT ML TECHNIQUES

There are three general frameworks in ML: supervised learning, unsupervised learning and reinforcement learning. These frameworks mainly differentiate on what data are sampled and how these sample data are used to build learning models. Under each framework, we will introduce several mainstream models. Sometimes, multiple learning models may work well for one given problem, and the appropriate selection can be made based on available hardware resources and data, implementation overheads, performance targets, etc.

2.1 Supervised Learning

Supervised learning is the process of learning a set of rules that are able to map an input to an output based on labeled datasets, where these learned rules can be generalized to make predictions for new, unseen inputs. According to the taxonomy of supervised learning [119], we provide a brief introduction to several prevalent models, as shown in Fig. 1.

(1) Decision trees, the representatives of logical learning methods, use tree structures to classify input instances, where each node represents a feature and branches of this node represent possible values of the corresponding feature. Starting from the root node, inputs are classified by sequentially passing nodes and branches, based on observed features and their values.

(2) Support vector machines (SVM) try to find the best hyperplanes to separate data classes by maximizing margins. Predictions or classifications of new inputs can be decided by their relative positions to these hyperplanes.

(3) Bayesian networks, one well-known representative of statistical learning algorithms, take the form of direct acyclic graphs (DAGs) to represent probability relationships among a set of random variables (i.e. features). In the DAG, each vertex denotes a random variable; each (directed) edge encodes the causal influences between the pair of random variables, while the absence of edge between variables indicates conditional independence.

(4) Artificial neural networks (ANNs) are capable to approximate a broad family of functions. With inspirations from neuroscience, ANNs employ collections of artificial neurons, and connect these neurons with learned weights, enabling particular neurons more sensitive to certain types of features. The versatility of ANNs to handle different learning tasks attributes to their various neural network structures: a single-layer perceptron is usually used for linear regression; complex DNNs consisting of multiple layers are able to approximate non-linear functions, such as the multi-layer perceptron (MLP); variants of DNNs that achieve excellent performance in specific fields benefit from the utilization of certain computation operations, e.g., convolutional neural networks (CNNs) with convolution operations taking advantage of spatial features, and recurrent neural networks (RNNs) with recurrent connections enabling learning from sequences and histories.

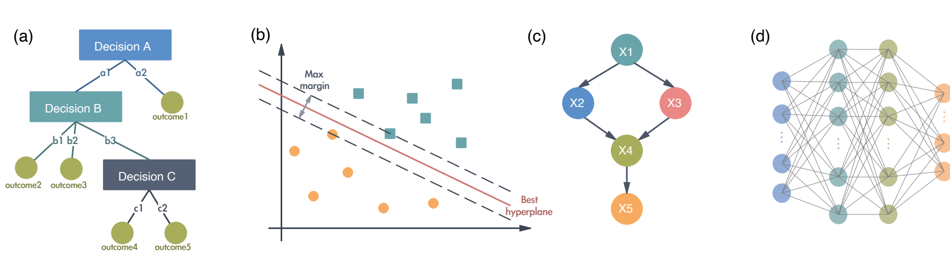

Fig. 1. Examples of supervised learning: (a) a decision tree, (b) a SVM, (c) a Bayesian network, and (d) an ANN.

Different learning models have different preference of input features: SVMs and ANNs generally perform much better with multi-dimension and continuous features, while logic-based systems tend to perform better when dealing with discrete/categorical features. In system design, supervised learning is commonly used for performance modeling, configuration predictions, or predicting higher level features/behaviors from lower level features, due to its great capability of function approximation and classification. One major bottleneck is the demand to create labeled training data prior to the training phase, indicating the necessity of human expertise and engineering; and in the meantime strongly labeled datasets are often laborious and expensive to obtain.

2.2 Unsupervised Learning

Unsupervised learning is the process of finding previously unknown patterns based on unlabeled datasets. Two prevailing methods are clustering analysis [94] and principal component analysis (PCA) [238], as depicted in Fig. 2.

(1) Clustering is a process of grouping data objects into disjoint clusters based on a measure of similarity, such that data objects in the same cluster are similar while data objects in different clusters share low similarities. The goal of clustering is to classify raw data reasonably and find possibly existing hidden structures or patterns in datasets, One of the most popular and simple clustering algorithms is k-means clustering [170].

(2) PCA is essentially a coordinate transformation leveraging information from data statistics. It aims to reduce the dimensionality of the high-dimensional variable space by representing it with a few orthogonal (linearly uncorrelated) variables that capture most of its variability.

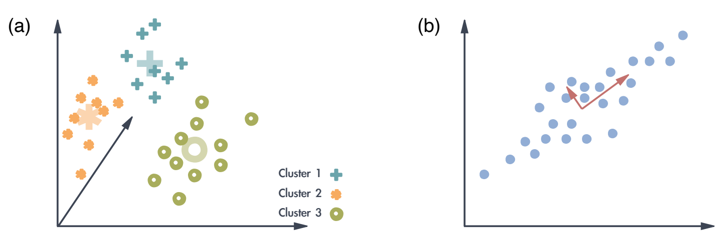

Fig. 2. Two prevalent methods in unsupervised learning: (a) clustering, and (b) PCA.

Since there are no labels in unsupervised learning, it is difficult to both measure the performance of learning models and decide when to stop the learning process. One noteworthy approach is *semi-supervised learning* [263], which uses a small amount of labeled data together with a large amount of unlabeled data. This approach stands between unsupervised and supervised learning, requiring less human effort and producing higher accuracy. The unlabeled data are used to either finetune or re-prioritize hypotheses obtained from labeled data alone. Several common models include expectation-maximization (EM) with generative mixture models, transductive learning, etc.

2.3 Reinforcement Learning

In standard reinforcement learning (RL) [211], an agent interacts with an environment $\mathcal{E}$ over a number of discrete time steps, as shown in Fig. 3. At each time step t, the agent receives a state s_t from the *state space* $\mathcal{S}$, and selects an action a_t from the *action space* $\mathcal{A}$ according to its policy π, where π is a mapping from states s_t to actions a_t. In return, the agent receives the next state s_{t+1} and a scalar reward $r_t : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$. This process continues until the agent reaches a terminal state after which the process restarts. The return $R_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k}$ is the total accumulated rewards at

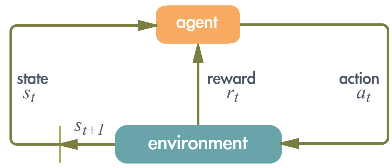

Fig. 3. A typical framing of RL.

the time step t with a *discount factor* $\gamma \in (0, 1]$. The goal of the agent is to maximize the expected return for each state s.

The state-action value $Q_\pi(s, a) = \mathbb{E}_\pi[R_t|s_t = s, a_t = a]$ is the expected return of selecting action a at state s with policy π. Similarly, the state value $V_\pi(s) = \mathbb{E}_\pi[R_t|s_t = s]$ is the expected return starting from state s by following policy π. There are two general types of methods in RL: value-based, and policy-based.

(1) In value-based RL, the state-action value function $Q_\pi(s, a)$ is approximated by either tabular approaches or function approximations. At each state s_t, the agent is always selecting the optimal action a_t^* that could bring the maximal state-action value $Q_\pi(s_t, a_t^*)$. One well-known example of value-based methods is Q-learning [235].

(2) In policy-based RL, it directly parameterizes the policy $\pi(a|s;\theta)$ and updates the parameters θ by performing gradient ascent on $\mathbb{E}[R_t]$. One example is the REINFORCE algorithm [237]. In standard REINFORCE, the policy parameters θ are updated in the direction of $\nabla_\theta \log \pi(a_t|s_t;\theta)R_t$, which is an unbiased estimate of $\nabla_\theta \mathbb{E}[R_t]$.

RL is modeled based on Markov decision process, and thus it is suitable to handle control problems or sequential decision-making processes. With these characteristics, RL is able to explore design space proactively and intelligently, and learn how to achieve resource management or task scheduling in system designs through interactions with environments. The optimal behaviors can be found by embedding optimization goals into reward functions.

3 ML FOR SYSTEM MODELLING

This section reviews studies that employ ML techniques for system modelling, which involve predictions of performance metrics or some other criteria of interest. Although cycle-accurate simulators, which are commonly used in system performance prediction, can provide accurate estimations, they usually run multiple orders of magnitude slower than native executions. In contrast, ML-based techniques are capable to balance the simulation cost and prediction accuracy, showing great potentials in exploring huge configuration spaces and learning non-linear impacts of configurations. Among most of existing work, supervised learning is widely applied, for either pure system modelling or efficient design space exploration enabled by fast predictions. We discuss these studies with respect to different levels of system that they are targeting, from circuit analysis, sub-systems to the system level.

3.1 Circuit Analysis

Circuit design is usually a manual process that requires many trial-and-error iterations between the pre-layout and post-layout phases, since subtle changes in the pre-layout phases can cause large impacts to circuit performance in an intricate manner. One effective way to circumvent the large number of iterations is to adopt performance modeling during the design flow. Conventional circuit performance modelling methods have high simulation costs, and this situation is exacerbated with the increasing complexity of integrated circuits, and the advent of newer process nodes.

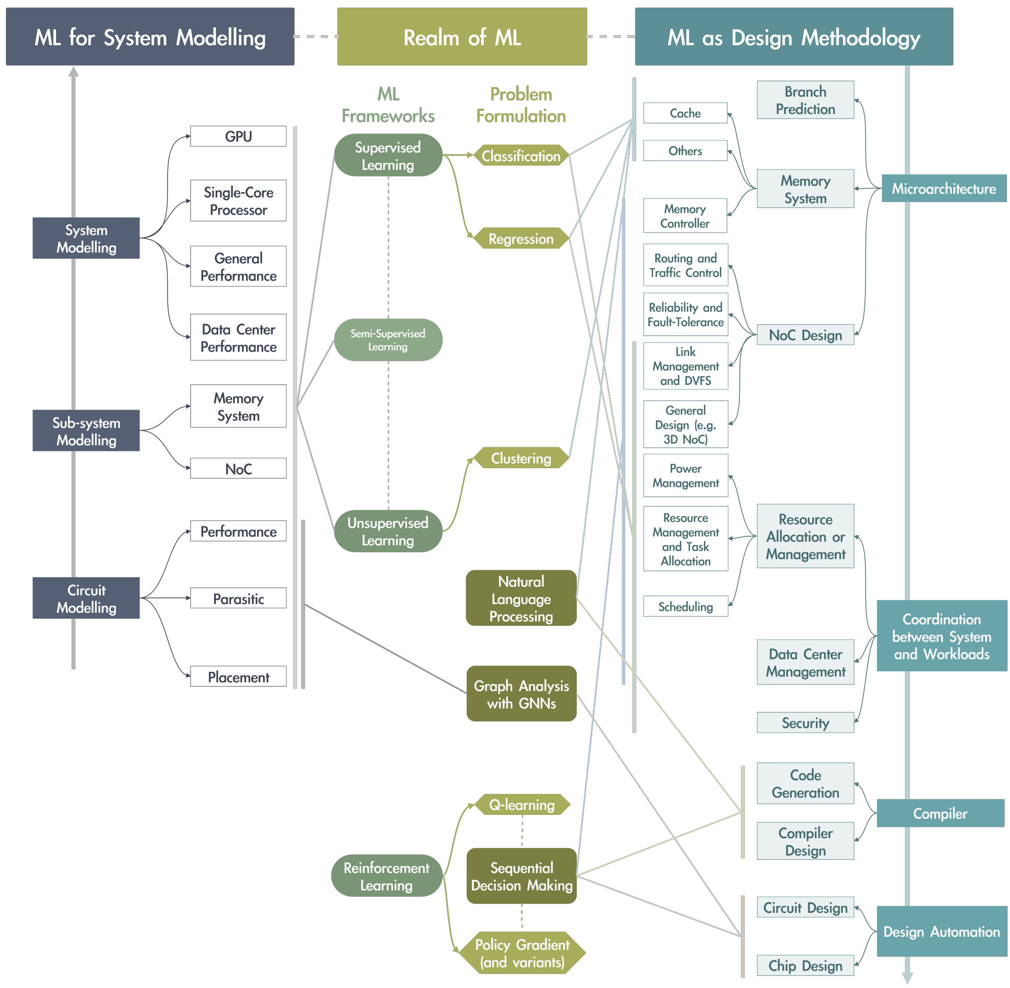

Fig. 4. A comprehensive overview of applying ML for computer architecture and systems. Existing work roughly falls into two categories: ML for system modelling, and ML as design methodology. Different system problems can be formulated as different ML problems. Natural language processing, graph analysis with GNNs and sequential decision making problems may span across multiple ML frameworks.

To leverage the hierarchical structure of integrated circuits, Alawieh *et al.* [6] propose a hierarchical performance modeling technique based on semi-supervised learning, which takes advantage of the Bayesian co-learning framework. This technique can generate pseudo samples from a large amount of unlabeled data, demonstrating the feasibility of performance modelling with inadequate labeled samples.

To leverage the inherent graph structures of circuit schematics, circuits can be modeled as graphs, and their related modelling in the pre-layout phases can be resolved by graph neural networks (GNNs). ParaGraph [190] builds a GNN model to predict layout-dependent parasitics and physical device parameters, and uses the ensemble modeling to further improve prediction accuracy. MLParest [205] shows that non-graph based method (e.g., random forest) can be used to estimate interconnect parasitics, whereas the lack of placement information may cause large variations in predictions. PEA [137] focuses on how circuit placement affects its performance, which problem is formulated as a classification problem. A customized GNN model, which can transfer knowledge across different topologies of the same circuit type, takes a placement solution as input, and predicts whether the post-routing performance meets certain specifications.

3.2 Sub-System Modelling and Performance Prediction

3.2.1 Memory Systems. In memory systems, especially non-volatile memories (NVMs), many efforts have been done to achieve different trade-offs between lifetime, performance and energy efficiency. To efficiently explore NVM-based cache hierarchies, Dong *et al.* [56] propose a circuit-architecture co-optimization framework that uses an ANN to predict higher level features (e.g. miss of cache read/write, and instruction-per-cycle (IPC)) from lower-level features (e.g. cache associativity, capacity and latency). To adaptively select architectural techniques in NVMs for different applications and objectives, Memory Cocktail Therapy (MCT) [52] estimates lifetime, IPC, and energy consumption through lighweight online predictors by gradient boosting and quadratic regression with lasso. MCT also conducts a comparison of different ML techniques regarding prediction accuracies, computation overheads, etc. To optimize placements of memory controllers in throughput processors, Lin *et al.* [141] build a DNN model to provide fast performance predictions, which takes memory controller placements and several features as inputs and uses multiple convolutional layers to analyze spatial localities. With expedited performance predictions, the search progress of optimizing memory controllers placements achieves speedup by two orders of magnitude.

3.2.2 Network-on-Chip (NoC). In NoCs, several performance metrics of interest are latency, energy consumption, and reliability. As to latency predictions, Qian *et al.* [184] use a support vector regression (SVR) model to predict the traffic flow latency and the average channel waiting time in mesh-based NoCs, which relaxes some assumptions in classical queuing theory. One major cause that deteriorates the average communication latency is the traffic hotspot, an intensive form of network congestion that significantly degrades the effective throughput of an NoC. There is a lightweight hardware-based ANN to predict hotspots in 2D-mesh NoC [111], using the buffer utilization rates from neighboring NoC routers to monitor the formation of hotspots. This ANN is trained by synthetic traffic pattern data offline, and evaluated using both synthetic and real application traces, achieving accuracy ranging from 65% to 92%. Their following work [208] combines this predictor with a proactive hotspot-preventive routing algorithm to avert hotspot formations, attaining significant improvements for synthetic workloads while modest melioration for real-world benchmarks.

As to energy consumption modelling, a few related criteria are often predicted. Aiming to save dynamic energy in NoCs, Clark *et al.* [36] leverage several ridge regression models to predict buffer utilization, changes in buffer utilization, or a combined metric of energy and throughput, based on which the router can select proper voltage/frequency. Similarly, Winkle *et al.* [221] also uses the ridge regression to predict the number of packets to be injected into each router in the following time window, under the scenario of photonic NoCs. With these predictions, they can scale the number of wavelengths and thus reduce the static energy consumed by photonic links. When considering both static and dynamic energy, DiTomaso *et al.* [55] use per-router decision trees to predict link utilization and traffic direction, which are then combined with sleepy link storage units to power-gate links/routers with low utilization and change link directions.

As to reliability of NoCs, which has become an issue in view of technology scaling down, aging, soft errors, process-voltage-temperature (PVT) variations, etc., there is a per-link decision tree trained offline to predict timing faults on links during runtime [54]. Equipped with these fault predictions, a proactive fault-tolerant technique is developed to mitigate errors, using the strengthened cyclic redundancy check with error-correction codes and relaxed transmission.

3.3 System Modelling and Performance Prediction

Accurate and fast performance estimation is a necessity for system optimization and design space exploration. With the increasing complexity of systems and variety of workloads, ML-based techniques, which have great generalization abilities, can provide high accuracy performance estimations with reasonable simulation costs, surpassing the capability of commonly-used cycle-accurate simulators that require high computational costs and long simulation time.

3.3.1 Graphics Processing Unit (GPU). There are two types of predictions for GPU modelling: cross-platform predictions and GPU-specific predictions. Cross-platform predictions aim to decide in advance whether to offload an application from a CPU to a GPU, since not every application benefits from GPU execution and the porting process requires significantly additional efforts; GPU-specific predictions are often used to model the performance of interest and assist GPU design space exploration, helpful to handle high irregularities of the design space and complicated interactions between configurations.

Cross-platform predictions can be made by using either dynamic program properties from execution or static analysis from code or intermediate representations. Using dynamic instruction profiles, Baldini *et al.* [13] formulate this as a binary classification problem to identify whether an application would achieve a GPU speedup over a threshold, where multiple supervised learning algorithms (i.e., nearest neighbor with generalized exemplars and SVMs) are tried. Using both both dynamic and static program properties from single-threaded CPU code, Ardalani *et al.* [8] predicts the GPU execution time through an ensemble of one hundred regression-based learners. Using merely static analysis of the source CPU code, their later work [9] employs a random forest composing of one thousand decision trees to make binary predictions that whether the potential speedup is greater than a given threshold.

GPU-specific predictions consist of both application-specific and general predictions. Among application-specific performance predictions, Stargazer [96] uses a stepwise regression modeling, which can recognize the most important parameters so as to achieve high prediction accuracy even with sparse and random samples, i.e., less than 3.8% average prediction error with 300 sampled design points in a design space with nearly 1 million possibilities. Jooya *et al.* [105] train multiple NN predictors and select the subset with the best generalization abilities to form an ensemble; with these performance/power predictions they further perform the Pareto-optimal multi-objective optimization. Among general predictions for performance of interest in GPUs, Wu *et al.* [240] model scaling behaviors of general-purpose GPUs (GPGPUs). They group training kernels with similar performance scaling behaviors by k-means clustering, and then build an ANN-based classifier to map a new kernel to the cluster that describes its scaling performance most properly. O'Neal *et al.* [174] predict cross-generation GPU performance by using an ensemble of 12 linear and 1 non-linear regression models. They use profiling results from earlier-generation GPUs (Haswell GT2) to train performance predictors for later/future-generation GPUs (Broadwell GT2/3, Skylake GT3), with more than 10,000 speedup compared to cycle-accurate GPU simulators. Li *et al.* [138] reassess prevailing assumptions of GPGPU traffic patterns, and propose a scheme that combines a CNN with a t-distributed stochastic neighbor embedding to classify different traffic patterns.

3.3.2 Single-Core Processor. In predictive performance modeling of single-core processors, early-stage work mostly targets superscalar processors. To predict the application-specific cycle-per-instruction (CPI) of superscalar processors, Joseph *et al.* [106] introduce an iterative procedure to build linear regression models using 26 key micro-architectural parameters. Later they construct predictive models by non-linear regression techniques (i.e., radial basis function networks generated from regression trees) with 9 key micro-architectural parameters [107]. They compare non-linear models with their linear counterparts, and experiment results indicate that the non-linear models can achieve better prediction accuracy (2.8% prediction error on average). In parallel with Joseph's work, Lee and Brooks [130, 132] use regression modeling with cubic splines to predict application-specific performance (billions of instructions per second) and regional power.

Later work focuses on performance modelling of existing hardware (e.g., Intel, AMD and ARM processors). Eyerman *et al.* [62] construct mechanistic-empirical models for CPI predictions of three Intel processors (i.e., Pentium 4, Core 2 and Core i7), with average prediction errors around 9% to 13%. These models are generated from mechanics where parameters are derived by regressions, and thus benefit from both mechanistic modeling (i.e., interpretability) and empirical modeling (i.e., ease of implementation). Zheng *et al.* [259, 260] explore two approaches to cross-platform predictions of program execution time, where program profiling results on Intel Core i7 and AMD Phenom processors are used to estimate the execution time on a target ARM processor. The first one [260] relaxes the assumption of global linearity to local linearity in the feature space, to apply constrained locally sparse linear regression; the other one [259] applies lasso linear regression with phase-level performance features.

3.3.3 General Modelling and Performance Prediction. Regression-based techniques are the mainstream to predict performance metrics from micro-architectural parameters or other features, thanks to their capability to make high-accuracy estimations by merely sampling a small subset of the large design space.

For regression-based models, ANN and (non-)linear regression with different designs are the common practice. Ipek *et al.* [89] use an ensemble of ANNs to predict IPC. Similarly, Khan *et al.* [117] employ an ANN to predict program execution time and the energy-delay product in chip-multiprocessor (CMP) systems. Lee and Brooks *et al.* use regression-based techniques with restricted cubic splines to predict Pareto frontiers in the power-delay space [129, 131]; they also propose the composable performance regression (CPR) [134], a hierarchical method estimating the multi-processor performance by combining baseline performance of each core and interference from other cores. Wu *et al.* [244] present strategies for integrated hardware-software performance predictions based on regression, where effective model specifications are constructed by the genetic search.Mantis [126] is an automatic performance modeling framework for Android applications on smartphones, which builds performance predictors by sparse non-linear regression using program-execution features with program slicing.

The are several comparisons among different regression techniques. Lee *et al.* [133] compare the piecewise polynomial regression with ANNs, with emphasis that conventional regression-based methods offer better explainability while ANNs have better generalization ability. Ozisikyilmaz *et al.* [175] make comparisons with respect to several methods for creating linear regression models and different types of ANNs, indicating that pruned ANNs achieve best accuracy though requiring longer training time. Agarwal *et al.* [4] estimate parallel execution speedups of multi-threaded applications on a target hardware, by using features and statistics extracted from the single-threaded execution. They also explore different learning-based methods and find that Gaussian process regression performs the best in their case.

More recent work tends to utilize data-driven approaches in regression-based systems. Ithemal [158] leverages a hierarchical multiscale RNN with long short term memory (LSTM) to predict throughput of basic blocks, where basic blocks are referred as sequences of instructions with no branches or jumps. Evaluations are conducted against two analytical throughput estimators, IACA [88] from Intel and llvm-mca [17] from LLVM, demonstrating that Ithemal is more accurate and as fast as these analytical tools. By employing a variant of Ithemal [158] as a differentiable surrogate to approximate original CPU simulators, DiffTune [191] is able to apply gradient-based optimization techniques to learn the parameters of x86 basic block CPU simulators such that the simulator's error is minimized, even within non-differentiable programs. The learned parameters finally are plugged back into the original simulator. Ding *et al.* [53] give some insights in learning-based modeling methods: the improvement of prediction accuracy may receive diminishing returns; it will be helpful to consider domain knowledge for system optimizations, even if the overall accuracy may not be improved. To this end, they propose to use a generative model to handle data scarcity by generating more training data, and apply a multi-phase sampling to improve prediction accuracy.

ML-based predictive performance modeling enables efficient resource management and rapid design space exploration to improve throughput. Bitirgen, Martinez and Ipek [19, 155] develop a framework for resource management in CMP, in which allocation decisions are made based on IPC predicted by an ensemble of ANNs. Likewise, equipped with an ANN for IPC predictions, Nemirovsky *et al.* [172] design a task scheduling policy to maximize system throughput in heterogeneous CPUs, which always selects the scheduling that would bring the best predicted IPC. ESP [164] constructs the regression model with elastic-net regularization to predict application interference (i.e., slowdown), which is integrated with schedulers to increase throughput. In consideration of rapid design space exploration of the uncore (i.e., both memory hierarchies and NoCs), Sangaiah *et al.* [196] uses a regression-based model with restricted cubic splines to estimate the CPI of CMP, reducing the exploration time by up to four orders of magnitude.

ML-based predictive performance modeling benefits adaptation of the trade-off between performance and certain power constraints or budgets. Curtis-Maury *et al.* [43–45] take advantage of different predictive performance models such as off-line (multivariate) linear regression models and ANNs, aiming to maximize performance of OpenMP applications in a power-aware manner by dynamic concurrency throttling (DCT) and dynamic voltage and frequency scaling (DVFS). A similar method [11] uses kernel clustering and off-line multivariate linear regression to predict power and performance of different applications, which is combined with hardware frequency-limiting techniques to select optimal hardware configurations under given power constraints. To effectively and efficiently apply DVFS towards various optimization goals, the corresponding strategy can adopt predictions for either power consumption by a constrained-posynomial model [109] or job execution time by a linear regression model [145]. To conduct smart power management in a more general manner, LEO [165] employs probabilistic graphical models (i.e., hierarchical Bayesian models) to predict performance and power, and when integrated for runtime energy optimization, it is capable to figure out the performance-power Pareto frontier and select the configuration satisfying the performance constraint with minimized energy. CALOREE [163] further breaks up the power management task into two abstractions: a learner for performance modelling and an adaptive controller leveraging predictions from the learner. These abstractions enable both the learner to use multiple ML techniques and the controller to maintain control-theoretic formal guarantees. As no user-specified parameter except the goal is required, CALOREE is applicable even for non-experts.

3.3.4 Data Center Performance Modelling and Prediction. Data centers have been widespreadly applied, from traditional enterprise applications to a variety of cloud services. With the increasing demand of data centers, several performance-wise issues arise, including optimization in the design space, improvement of resource utilization, etc. In view of these issues, majority studies predict job/task length, resource demand, workload pattern, and related performance metrics, for configuration auto-tuning or elastic resource provisioning purposes. These predictions must be completed in advance so that the management system can either tune the configurations or adjust resource allocations ahead of the needs.

For job/task length prediction, Ganapathi *et al.* [68] propose to predict several performance metrics (e.g. the actual elapsed times, disk I/Os, etc.) of database queries, where the model is trained by kernel canonical correlation analysis (KCCA) and makes predictions based on information from the m-nearest neighbors. Yigitbasi *et al.* [250] use SVR to predict the job completion time of Hadoop MapReduce applications on different clusters by using both application characteristics and cluster configurations. For resource demand prediction, light-weight statistical learning algorithms can be leveraged to predict dynamic resource demands of both cyclic and non-cyclic workloads [75], which achieve good prediction accuracy with less than 5% over-estimation error and near zero under-estimation error. The upcoming resource demands can be predicted by using MLP or linear regression [91]. For workload prediction/forecasting, a second order autoregressive moving average (ARMA) method can estimate incoming workloads of the system for future time periods [193]. Its generalization, the autoregressive integrated moving average (ARIMA) model, is able to serve cloud workload forecasting [27]. To predict variations of workload patterns, the hidden Markov modeling (HMM) can be used to characterize the temporal correlations in clusters [116].

Some work has been evaluated in commercial data centers. Jim Gao [69] builds an MLP model to predict power usage effectiveness (PUE) of data centers, which is extensively tested and validated at Google data centers. Cortez *et al.* [40] predict several virtual machine (VM) behaviors (including VM lifetimes, maximum deployment sizes, and workload classes) for a broader set of purposes (including health management and power capping). They introduce the Resource Central (RC), a system that ingests VM telemetry, periodically learns VM behaviors offline, and provides predictions online to various resource management systems. In their design, RC does not automatically select the proper ML modeling approach, leaving this task for experts. To demonstrate RC's capability, they use random forests and extreme gradient boosting trees for metrics modelling, and modify Microsoft Azure's VM scheduler to leverage predictions in oversubscribing servers, which increases resource utilization and prevents physical resource exhaustion.

4 ML AS DESIGN METHODOLOGY

This section introduces how ML can be employed as a design methodology to directly enhance architecture/system designs. Computer architecture and systems have been becoming increasingly complicated, making it prohibitively expensive and inefficient for human efforts to either design or optimize them. In response, visionaries have argued that computer architecture and systems should be imbued with the capability to design and configure themselves, adjust their behaviors according to workloads' needs or user-specified constraints, diagnose failures, repair themselves from the detected failures, etc. With strong learning and generalization capabilities, ML-based techniques are naturally suitable to resolve these considerations, which can adjust their policies according to long-term planning and dynamic workload behaviors during system designs.

4.1 Memory System Design

The "memory wall" has been a performance bottleneck in von Neumann architectures for many years, where the computation is orders of magnitude faster than the memory access. To alleviate this problem, hierarchical memory systems are widely used and there arise optimizations for different levels of memory systems. As both the variety and the size of modern workloads are drastically growing, conventional memory system designs that are based on heuristics or intuitions can not catch up with the demand of these ever-growing workloads, leading to sharply degradation in system performance. Additionally, the scalability is another concern in these heuristic-based designs. In contrast, ML provides promising potentials, whose generalization ability naturally addresses the scalability issue. By performing analogies from memory address predictions to label predictions, and from memory access sequences analysis to sequence predictions in natural language process, ML can become a powerful tool to optimize performance of memory systems.

4.1.1 Cache. The conspicuous disparity in latency and bandwidth between CPUs and memory systems motivates investigations of efficient cache management. There are two major types of studies on cache optimization: improving cache replacement policies, and designing intelligent prefetching policies.

To develop better cache replacement policies, Teran *et al.* [217] use perceptron learning to predict whether to bypass or reuse a referenced block in the last-level cache (LLC). Their following work, multiperspective reuse prediction [102], achieves further improvements in cache performance by employing multiple types of features and considering both the reuse information and placement positions of the referenced block. Instead of using perceptrons, Beckmann *et al.* [14] model the cache replacement problem as a Markov decision process and adopt the idea of replacing lines according to their economic value added (EVA), i.e., the difference between their expected hits and the average hit. Shi *et al.* [201] train an attention-based LSTM model offline to extract informative insights from history program counters (PCs), which are then used to build an online SVM-based hardware predictor to form their "Glider" cache replacement policy.

To devise intelligent prefetchers, there are studies ranging from tuning configurations to improving their policies. With regard to optimizing configurations, program characterizations and hardware performance counters can be used to predict whether to enable prefetchers at different cache levels [185], and there is a in-depth comparison [139] among multiple ML models regarding different benchmarks. With regard to designing prefetching policies, Wang *et al.* [228] propose a prefetching mechanism that uses conventionally table-based prefetchers to provide prefetching suggestions and a perceptron trained by spatio-temporal locality to reject unnecessary prefetching decisions, ameliorating the cache pollution problem. Similarly, Bhatia *et al.* [16] integrate perceptron-based prefetch filtering with conventional prefetchers, increasing the coverage of prefetches without hurting accuracy. Instead of the commonly used spatio-temporal locality, a context-based memory prefetcher [182] leverages the semantic locality that characterizes access correlations inherent to program semantics and data structures to prefetch data blocks accordingly, which is approximated by the contextual bandits model in RL. Interpreting and understanding semantics in memory access patterns are analogous to sequence analysis in natural language processing (NLP), and thus several studies use LSTM-based models and treat the prefetching as either a regression problem [255] or a classification problem [80]. Even with better performance, especially for long access sequences and noise traces, the LSTM-based prefetcher suffers from long warm-up and prediction latency, and considerable storage overheads. The discussion of how hyperparameters impact LSTM-based prefetcher performance [24] highlights that the lookback size (i.e. memory access history window) and the LSTM model size strongly affect the prefetcher learning ability under different noise levels or workload patterns. To accommodate the large memory space, Shi *et al.* [202] introduce a neural hierarchical sequence model to decouple predictions of pages and offsets by using two separate attention-based LSTM layers, whereas its hardware implementation is impractical for actual processors.

4.1.2 Memory Controller. Smart memory controllers can further improve memory bandwidth utilization. Aiming at a self-optimizing memory controller that is adaptive to dynamically changing workloads [90, 155], the memory controller is modeled as an RL agent that always selects legal DRAM commands with the highest expected long-term performance benefits (i.e., Q-values). To allow optimizations toward various objectives, this memory controller is improved in two major aspects [168]. First, the rewards of different actions (i.e., legal DRAM commands) are automatically calibrated by genetic algorithms to serve different objective functions (e.g.,energy, throughput, etc). Then, a multi-factor method that considers the first-order attribute interactions is employed to select attributes used for state representations. Since both of them use table-based Q-learning and select limited attributes to represent states, the scalability may be a concern and their performance could be improved with more informative representations.

4.1.3 Others. Instead of focusing on a specific object, some researchers consider more general issues. For example, early-stage work predicts the repetitive memory access patterns of parallel scientific applications on multiprocessors with several trainable techniques [194]. From the storage side, Block2Vec [46] tries to mine disk block correlations by training a DNN to learn the best multi-dimensional vector representation of each block and capturing block similarities via vector distances, which enables further optimizations for caching, prefetching, etc. From the program side, Shi *et al.* [203] use a GNN to learn fused representations of the static code and its dynamic execution. This unified representation is capable to model both the data-flow (i.e., prefetching) and the control-flow (i.e., branch prediction).

A variety of work targets different parts of the memory system. Margaritov *et al.* [154] accelerate virtual address translation through learned index structures [120]. The results are encouraging in terms of the accuracy that reaches almost 100% for all tested virtual addresses, yet still with unacceptably long inference latency, leaving practical hardware implementation as the future work. Wang *et al.* [233] reduce data movement energy in interconnects by exploiting asymmetric transmission costs of different bits, in which transmitted data blocks are dynamically grouped by the k-majority clustering to derive energy-efficient expressions for transmission. In terms of garbage collection in NAND flash, Kang *et al.* [112] propose an RL-based method to reduce the long-tail latency. The key idea is to exploit the inter-request interval (idle time) to dynamically decide the number of pages to be copied or whether to perform an erase operation, and decisions are made by the table-based Q-learning. Their following work [113] considers more fine-grained states, and introduces the Q-table cache to manage key states among enormous amount of states.

4.2 Branch Prediction

Branch predictor is one of the mainstays of modern processors, significantly improving the instruction-level parallelism. As pipelines gradually deepen, the penalty of mis-prediction increases. Traditional branch predictors often consider limited history length, which may hurt the prediction accuracy. In contrast, the perceptron/MLP-based predictors can handle long histories with reasonable hardware budgets, outperforming prior state-of-the-art non-ML-based predictors.

Starting with a static branch predictor that is trained with static features from program corpus and control flow graphs [26], it employs an MLP to predict the direction of the branch at compile time. Later, a dynamic branch predictor [101] uses a perceptron-based method, which hashes the branch address to select the proper perceptron and computes the dot product accordingly to decide whether to take this branch, showing great performance on linearly separable branches. Its latency and accuracy can be further improved by applying ahead pipelining and selecting perceptrons based on path history [97]. To attain high accuracy in non-linearly separable branches, the perceptron-based prediction is generalized as piecewise linear branch prediction [98]. In addition to the path history that is used in the above work, multiple types of features from different organizations of branch histories can be leveraged to enhance the overall performance [100]. SNAP [209] proposed a practical hardware implementation, which makes use of current-steering digital-to-analog converters to transfer digital weights into analog currents and replaces the costly digital dot-product computation to the current summation. Its optimized version, OH-SNAP [99], equips several new techniques such as the use of global and per-branch history, trainable scaling coefficients, dynamic training thresholds, branch cache, etc.

Rather than making binary decisions of whether to take a certain branch, perceptron-based predictors [72] can directly predict the target address of an indirect branch at the bit level. Even though high accuracy is achieved by current perceptron/MLP-based predictors, Tarsa *et al.* [216] notice that a small amount of static branch instructions are systematically mispredicted, referred to as hard-to-predict branches (H2Ps). Consequently, they propose CNN helper predictors that encode history branches to form history matrices and leverage a CNN to take advantage of pattern matching, ultimately improving accuracy for H2Ps in conditional branches.

4.3 NoC Design

The aggressive transistor scaling has paved the way for integrating more cores in a single chip or processor. With the increasing number of cores per chip, NoC, which is responsible for both inter-core communication and data movement between cores and memory hierarchies, plays a gradually crucial role. There are several emerging problems attracting attention. First, the communication energy scales slower than the computation energy [21], implying necessity to improve power efficiency of NoCs. This is a challenging problem, especially in those heterogeneous multi-core or many-core systems. Second, the complexity of routing or traffic control grows with the number of cores per chip and this problem is even exacerbated by the rising variety and irregularity of workloads. Third, with the continuous scaling down of transistors, NoCs are more vulnerable to different types of errors and thus reliability becomes a key concern. Last but not the least, some non-conventional NoC architectures might bring promising potentials in the future, while they usually come with large design spaces and complex design constraints to comply, which is nearly impossible for manually optimization. Among all these fields, the ML-based design techniques display their strength and charm.

4.3.1 Link Management and DVFS. Power consumption is one crucial concern in NoCs, in which links usually consume a considerable portion of network power. While turning on/off links according to a static threshold of link utilization is a trivial way to reduce power consumption, it can not adapt to dynamically changing workloads. Savva *et al.* [197] use ANNs for dynamic link management, where each ANN is responsible for one region of the NoC and dynamically computes a threshold for each time interval to turn on/off each link by using the link utilization of each region. Despite significant power savings with low hardware overheads, this approach causes long latency in routing. In order to meet certain power and thermal budgets, hierarchical ANNs [192] are used to predict optimal NoC configurations (i.e., link bandwidth, node voltage and task assignment to nodes), where the global ANN predicts globally optimal NoC configurations exploiting local optimal energy consumption predicted by local ANNs. To save dynamic power, several investigations [65, 258] employ per-router based Q-learning agents that are built by offline trained ANNs to select optimal voltage/frequency levels for each router.

4.3.2 Routing and Traffic Control. With the increasing variety and irregularity of workloads and their traffic patterns, learning-based routing algorithms and traffic control approaches show superior performance due to their excellent adaptability. Since routing problems can be formulated as sequential decision making processes, suitable to the realm of RL, several studies apply Q-learning approaches, namely the Q-routing algorithm [23]. Q-routing uses local estimations of delivery time to minimize total packets delivery time, which is able to handle irregular network topologies and keep a higher network load than the conventional shortest path routing. It is then extended to several other scenarios, for example, combining with dual RL to improve both the learning speed and the routing performance [125], resolving packets routing in dynamic NoCs whose network structures/topologies are dynamically changing during runtime [152], handling irregular faults in bufferless NoCs by the reconfigurable fault-tolerant Q-routing [64], and enhancing the capability to reroute messages around congested regions by the congestion-aware non-minimal Q-routing [59]. In addition to routing problems, deep Q-network is also capable for NoC arbitration policies [251], where the agent/arbiter always grants a certain output port to the input buffer with the largest Q-value. The following work [252] thoroughly compares three reward functions (i.e., the global age of messages, the reciprocal of average accumulated latency, and NoC link utilization), among which the global age based reward function has better performance. Even displaying some improvements in both latency and throughput, the direct hardware implementation is impractical due to the complexity of deep Q-networks, thus from which they distill insights to derive a relatively simple circuitry implementation.

Adjusting injection rates is an efficient way to control congestion in NoCs. The SCEPTER NoC architecture [49], a bufferless NoC with single-cycle multi-hop traversals and a self-learning throttling mechanism, controls the injection of new flits into the network by Q-learning, where each node in the network independently selects whether to increase, decrease or retain the throttle rate according to their Q-values, conspicuously improving bandwidth allocation fairness and network throughput. Wang *et al.* [227] design an ANN-based admission controller to determine the appropriate injection rate and control policy of each node in a standard NoC.

4.3.3 Reliability and Fault Tolerance. With the aggressive technology scaling down, transistors and links in NoCs are more prone to different types of errors, indicating that reliability is a crucial concern and proactive fault-tolerant techniques are required to guarantee performance. Wang *et al.* [231] employ per-router Q-learning agents to independently select one of four fault-tolerant modes, minimizing the end-to-end packet latency and power consumption. These agents are pre-trained and then fine-tuned during runtime. In their following work [232], these error-correction modes are extended and combined with various multi-function adaptive channel configurations, retransmission settings and power management strategies, eventually reducing the latency, and improving the energy efficiency and mean-time-to-failure.

4.3.4 General Design. With the growing number of cores per chip/system, the increasing heterogeneity of cores and various performance targets, it is complicated to optimize NoC designs, which involves optimizing copious variables simultaneously. One attempt to the automatic NoC design flow is the MLNoC [186], which utilizes supervised learning to quickly find near-optimal NoC designs under multiple optimization goals. MLNoC is trained by data from thousands of real-world and synthetic SoC (system-on-chip) designs with a wide range of characteristics, and evaluated only with real-world SoC designs. Despite disclosure of limited details and absence of comprehensive comparison with other design methods, it shows superior performance to manually optimized NoC designs, delivering encouraging results.

Apart from conventional 2D mesh NoCs, a series of investigations focuses on designs of 3D NoCs. Das *et al.* [47, 48] apply the STAGE algorithm [22] to optimize both vertical and planar placement of communication links in small-world network based 3D NoCs. The STAGE algorithm repeatedly alternates between two stages, the base search that tries to find the local optima based on the original objective, and the meta search that uses SVR to learn evaluation functions. Later, the STAGE algorithm is extended for multi-objective optimization in heterogeneous 3D NoC systems [104], which jointly considers the GPU throughput, average latency between CPUs and LLCs, temperature and energy.

In terms of routerless NoCs that any two nodes are connected via at least one ring/loop, Lin *et al.* [142] develop a deep RL framework to optimize loop placements, with a Monte-Carlo tree search for efficient design space exploration. The RL agent leverages a deep convolutional neural network to approximate policy and value functions, and the design constraints can be strictly enforced by carefully devising the reward function.

4.4 Resource Allocation or Management

Resource allocation or management is the coordination between computer architecture/systems and workloads. Consequently, its optimization difficulty occurs with the booming complexity from both sides and their intricate interactions. ML-based approaches have blazed the trail to adjusting policies wisely and promptly pursuant to dynamic workloads or specified constraints, surpassing conventional techniques.

4.4.1 Power Management. ML-based techniques have been applied broadly to improve power management, due to two main reasons. First, power/energy consumption can be recognized as one metric of runtime costs. Second, under certain circumstances there could be a hard or soft constraint/budget of power/energy, making power efficiency a necessity.

Several investigations consider power management for different parts of systems. PACSL [1] uses a supervised learning technique, namely the propositional rule, to adjust dynamic voltage scaling (DVS) for CPU cores and the on-chip L2 cache. Results indicate an improvement in the energy-delay product by 22% on average (up to 46%) over independently applying DVS for each domain. Instead of focusing on CPU cores, Won *et al.* [239] coordinate an ANN controller with a proportional integral (PI) for uncore DVFS. The ANN controller can be either pre-trained offline by a prepared dataset or trained online by bootstrapped learning; once the ANN training phase is completed, tandem ANN-PI control operations are applied to better accommodate different workloads. Manoj *et al.* [181] deploy Q-learning to adaptively adjust the level of output-voltage swing at transmitters of 2.5D through-silicon interposer I/Os, under constraints of both communication power and bit error rate.

From the system level, DVFS is one of the most prevalent techniques. Pack & Cap [37, 189] builds a multinomial logistic regression (MLR) classifier, which is trained offline and queried during runtime, to accurately identify the optimal operating point for both thread packing and DVFS under an arbitrary power cap. GreenGPU [149] focuses on the heterogeneous systems with CPUs and GPUs, and applies the weighted majority algorithm [144] to scale the frequency levels for both GPU cores and memory in a coordinated manner. CHARSTAR [187] integrates the power gating with DVFS in a single-core processor, and employs a reconfiguration mechanism aware of the clock hierarchy, where the frequencies and configurations are dynamically predicted by a lightweight offline trained MLP predictor. To minimize energy consumption, Imes *et al.* [87] use ML-based classifiers to predict the most energy-efficient resource settings (specifically, tuning socket allocation, the use of HyperThreads, and processor DVFS) by using low-level hardware performance counters.

A series of studies leverages RL for dynamic power management in multi/many-core systems, since RL is excelled at sequential decision making and adjusting policies through continuous observations of workload dynamics. One possible issue is the scalability. As systems scale up, these RL-based methods often suffer from state space explosion, which could significantly impact the training costs. To this end, there are two main types of methods: combining RL with supervised learning, and hierarchical RL. Regarding the first type, a semi-supervised RL-based approach [110] achieves linear complexity with the number of cores, which is able to maximize throughput ensuring power constraints and cooperatively control cores and uncores in synergy. As for the second type, hierarchical Q-learning can reach the time complexity $O(n \lg n)$ with the number of cores. Multi-Level RL (MLRL) [177] is a scalable and effective online policy to select target power modes, where the Q-values are approximated by the generalized growing and pruning radial basis function. Similarly, table-based distributed Q-learning also performs well for DVFS [32], and there is one variant, Profit [33], aware of the priorities of different applications.

Some energy management policies target specific applications or platforms. JouleGuard [84] is a runtime control system coordinating approximate applications with system resource under energy budgets. It uses RL (i.e., the multi-armed bandit approach) to identify the most energy efficient system configuration, and given this configuration it further determines the application configuration satisfying the energy goal with maximized accuracy. Bai *et al.* [10] considers the on-chip regulator efficiency loss during DVFS, trying to minimize the energy under a parameterized performance constraint. The online control policy is implemented by a table-based Q-learning, portable across platforms without accurate modeling of a specific system. Tarsa *et al.* [215] present a post-silicon CPU customization based on Intel SkyLake, which applies various ML models to predict cluster gating for dynamically clock-gating unused resources.

4.4.2 Resource Management and Task Allocation. Modern architectures and systems have been becoming so sophisticated and diverse that it is non-trivial to either optimize performance or fully utilize system resources. This rapidly evolving landscape is further complicated by various workloads with specific requirements or targets. In order to keep the pace, it is a necessity to develop more efficient and automatic methods, which should be capable to tailor resources to specified requirements and adapt the hardware cost-effectively at runtime to applications' needs. ML-based techniques are skilled to explore extremely large design space and optimize multiple objectives simultaneously; with careful designs, they can preserve better scalablility and great portability to different platforms.

Starting from a single-core processor, a regularised maximum likelihood approach [58] is used to predict the best hardware micro-architectural configuration for each phase of a program, based on hardware counters collected at runtime. To identify optimized configurations in a multi-core processor, statistical machine learning (SML) based auto-tuning method [67] is able to quickly figure out configurations that simultaneously optimize running time and energy efficiency. This method is agnostic to application and micro-architecture domain knowledge, leading to a scalable and portable alternative to human expert-optimization. SML can also be used as a holistic method to design self-evolving systems that optimize performance hierarchically across the circuit, platform, and application levels [20].

In multi-core processors, dynamic on-chip resource management is crucial. One example is the dynamic cache partitioning. In response to the changing needs of jobs, L2 cache can be dynamically partitioned by an RNN that is evolved by the enforced subpopulations algorithm [74]. The dynamic partitioning of LLC can be further integrated with DVFS on the cores and uncore [95], using a table-based Q-learning as the co-optimization method, which results in much lower energy-delay product (EDP) than any of the techniques applied individually.

To guarantee efficient and reliable execution in many-core systems, task allocation should consider several aspects, such as heat and communication issues, where RL is often deemed as an effective resolution. Targeting the heat interaction of processor cores and NoC routers, Lu *et al.* [147] apply Q-learning to perform task assignments to specific cores based on current temperatures of cores and routers, such that the maximum temperature in the future is minimized. Targeting the non-uniform and hierarchical on/off-chip communication capability in multi-chip many-core systems, core placement optimization [242] leverages deep deterministic policy gradient (DDPG) [140] to map computation onto physical cores, able to work in a manner agnostic to domain-specific information.

For a broader view, there are workflow management and hardware resource assignment adopted in more general cases. SmartFlux [61] focuses on the workflow of data-intensive and continuous processing. It intelligently guides the asynchronous triggering of processing steps with the help of predictions made by multiple ML models, to indicate whether to execute certain steps and their corresponding configurations upon each wave of data. Given the target DNN model, the deployment scenario, platform constraints, and optimization objectives (latency/energy), ConfuciuX [114] applies a hybrid two-step scheme for optimal hardware resource assignments (i.e., assigning the number of processing elements and the buffer sizes to each DNN layer), where the REINFORCE [237] is used to perform a global coarse-grained search followed by a genetic algorithm for fine-grained tuning.

In heterogeneous systems with CPUs and GPUs, device placement refers to the process of mapping nodes in the computational graphs of NNs onto proper hardware devices. Mirhoseini *et al.* [162] propose an RL-based method (i.e., policy gradient via REINFORCE) for device placement optimization, which uses a sequence-to-sequence RNN model as the parameterized policy to generate placements. This work manually groups operations and then automatically places them on devices. Later, a hierarchical end-to-end model [160] is developed to make this manual grouping process automatic. Spotlight [71] employs the proximal policy optimization (PPO) to achieve better training speed and uses the softmax distributions to represent the policy. Their later work [70] integrates PPO with cross-entropy minimization to acquire theoretically guaranteed optimal efficiency. One thing worth noting is that these approaches are not transferable and a new policy should be specifically trained from scratch for each new and unseen computational graph. To get generalizable solutions, Placeto [2] uses graph embeddings to encode the structure of the computational graph and exhibits good generalizability to unseen neural networks, while having high computation costs. Generalize Device Placement (GDP) [261] makes use of a graph embedding network that learns graph embeddings of arbitrary dataflow graphs, and a placement network that uses a scalable sequential attention mechanism to learn a placement strategy given graph embeddings. These two components are trained jointly in an end-to-end manner over a set of dataflow graphs, and then fine-tuned for each specific graph, eventually reaching 15× faster convergence than prior arts [160, 162].

4.4.3 Scheduling. In classical real-time scheduling problems, the key task is to merely decide the orders, according to which the currently unscheduled jobs should be executed by a single processor, such that the overall performance is optimized. As multi-core processors have been the mainstream, the scheduling is gradually perplexing. One major reason is that multiple objectives besides the performance should be carefully considered, such as balanced assignments among various cores and response time fairness. Equipped with the capability to well understand the feedback provided by the environment and dynamically adjust policies, RL is a common tool for real-time scheduling.

To optimize the execution order of jobs after they are routed to a single CPU, the adaptive scheduling exploits Q-routing [236], where the proposed insertion scheduler utilizes the router's Q-table to assess a job's priority and decides jobs' ordering accordingly so as to maximize the overall utility. In multi-core systems, Fedorova *et al.* [63] present a blueprint for a self-tuning scheduling algorithm to maximize a performance function that is an arbitrary weighted sum of metrics of interests, which is built upon the value-based temporal-difference method in RL. This algorithm is further improved to be a general methodology for online scheduling of parallel jobs onto multi-processor systems [223], where the value functions are approximated by a parameterized fuzzy rulebase with temporal difference. This scheduling policy always selects jobs that have the maximum value functions from the job queue to execute, possibly preempting some currently running jobs and squeezing some jobs into fewer CPUs than they ideally require, while with optimized long-term utility.

Focusing on PIM-assisted GPU architectures, Pattnaik *et al.* [180] roughly categorize GPU cores into two types: the powerful GPU cores yet far away from memory, and the auxiliary/simple GPU cores yet close to memory. Two runtime techniques are proposed: first, a regression-based affinity prediction model is used to accurately identify which kernels would benefit from PIM and offload them accordingly to the auxiliary cores; then, a management mechanism is developed to decide which kernels can be scheduled concurrently on the two types of cores, integrating the affinity prediction model, a new regression-based execution time prediction model, and dependency information across kernels.

4.5 Data Center Management

With the rapid expansion of the scale of data centers, issues that may be trivial in a single machine become increasingly challenging, let alone the inherently complicated problems.

Early work aims at a relatively simple scenario of resource allocation, which is to dynamically assign different numbers of servers to multiple applications. This problem can be modeled as an RL problem with the service-level utility function as the reward: the arbiter will select a joint action that would bring the maximum total return after consulting local value functions that can be estimated via either table-based methods [219] or function approximation [218]. A similar approach is implemented and tested in Oracle Solaris 10 [225], which demonstrates a robust and near-optimal performance on transferring CPUs among resource partitions to match the stochastically changing workload. To better model interactions between multiple agents, a multi-agent coordination algorithm with fuzzy RL [224] can be used to solve the dynamic content allocation in content delivery networks (CDNs), in which each requested content is modeled as an agent, trying to move toward the area with a high demand while coordinating with other agents/contents.

From the aspect of design space exploration, tuning system configurations plays an indispensable role in improving performance. OtterTune [220] combines supervised and unsupervised learning methods for database management system configuration tuning. It prunes redundant metrics by feature analysis and k-means clustering, selects most important knobs by the lasso path algorithm, dynamically maps target workloads to the most similar known workloads, and finally recommends knob configurations by Gaussian process regression.

From the aspect of availability in data centers or cloud services, disk failure is one of the leading reasons of service unavailability. Leveraging SMART (Self-Monitoring, Analysis and Reporting Technology) attributes, many researchers can build disk failure prediction models via various ML techniques, such as different Bayesian methods [78], unsupervised clustering [169], SVM and MLP [262]. The adoption of either classification and regression trees (CART, i.e., decision trees) [135] or gradient boosted regression trees [136], can output both classification results and health assessments of drives. To explicitly exploit sequential information of SMART attributes, Xu *et al.* [246] use RNNs to classify drives into multiple levels to assess health status, according to their remaining lifetime. One thing worth noticing is that all the aforementioned methods rely on offline training, which impedes the adaptation to forthcoming data, thus suffering from the 'model aging' problem. Xiao *et al.* [245] propose to use online random forests (ORFs) to predict disk failures based on the SMART data, which evolve with forthcoming data on-the-fly by generating new trees and forget old information by discarding outdated trees, consequently avoiding the model aging problem. In addition to complete disk failures, another threat is the partial drive failure, i.e., disk error (e.g., sector error and latency error). Mahdisoltani *et al.* [151] explore five ML techniques (i.e., CART, random forests, SVM, NN and logistic regression) to predict sector errors, among which random forests consistently outperform others and the training process is robust to either small training sets or training data from a non-target system. For online disk error prediction, Cloud

Disk Error Forecasting (CDEF) [247] incorporates both SMART attributes and system-level signals to build a cost-sensitive ranking-based prediction model by using a multiple additive regression trees gradient boosting algorithm, which ranks disks according to the degree of error-proneness in the near future. Currently, CDEF is successfully applied in Microsoft Azure.

From the aspect of improving quality of experience (QoE) for users, it is essential to deploy an intelligent data center level cache. Toward a self-adaptive caching mechanism, DeepCache [171] employs the LSTM encoder-decoder model to predict future content popularity, which can be combined with existing cache policies to make smarter decisions. Phoebe [243] is an online caching framework leveraging DDPG, which targets a large variety of emerging storage models. Considering non-history based features, Wang *et al.* [230] build a decision tree to predict whether the requested file will be accessed only once inn the future. These one-time-access files will be directly sent to users without getting into cache, to avoid cache pollution. Traffic optimization is an alternative to improve QoE. Chen *et al.* [28] develop a two-level deep RL system: the peripheral systems that are trained by DDPG make instant traffic optimization decision locally for short flows; the central system that is trained by policy gradient aggregates global traffic information, guides behaviors of peripheral systems, and makes individual traffic optimization decisions for long flows.

From the aspect of of workloads, video workloads on CDNs or clusters are prevalent but their optimization is quite challenging: first, the network conditions fluctuate overtime and a variety of QoE goals should be balanced simultaneously; second, only coarse decisions are available and the current decisions will have long-term effects on following decisions. This scenario naturally matches the foundation of RL-based techniques. To optimize users' QoE of streaming videos, the adaptive bitrate (ABR) algorithms have been recognized as the primary tool used by content providers, which execute on client-side video players and dynamically choose a bitrate for each video chunk based on underlying network conditions. Pensieve [153] applies asynchronous advantage actor-critic [166] to select proper bitrate for future video chunks based on the resulting performance from past decisions. The following work [249] integrates an RL agent designed from Pensieve to decide whether to enhance video quality or use the video bitrate provided by Pensieve. When considering large-scale video workloads in hybrid CPU-GPU clusters, the performance degradation often comes from the uncertainty and variability of workloads, and the unbalanced use of heterogeneous resources. To accommodate this, Zhang *et al.* [256] use two deep Q-networks to build a a two-level task scheduler, where the cluster-level scheduler selects proper execution nodes for mutually independent video tasks and the node-level scheduler assigns interrelated video subtasks to appropriate computing units.

4.6 Security

Security issues include but not limited to the execution integrity and protection from malicious attacks. However, the evolving scale and complexity of computing systems often give rise to more proneness to faults and increasing variety of attacks, which has posed a challenge on the security side.

In a large-scale system, faults often occur intermittently and may come from each part of the system for a wide range of reasons. This requires diagnosis techniques to be able to reason out the key problem from numerous potential causes of failure in time. Given observations at a web server, Platt *et al.* [183] apply approximate Bayesian inference for failure diagnosis, which can quickly determine the failure part and accurately estimate its underlying failure rates.

The proliferation and evolvement of computing systems are usually followed by the proliferation and evolvement of malware. Malware detection is often modeled as a classification problem, to distinguish malicious programs from benign programs. The detection is feasible based on either hardware performance counters (HPCs) or software features. To build a robust hardware-based malware detector, standard supervised classification algorithms (e.g., k-Nearest Neighbors, decision trees, random forests, and MLP) [51] can take advantage of statistics from hardware performance counters (HPCs) to effectively detect variants of known malware in offline validation. Unsupervised learning techniques, such as the one-class SVM classifier that uses the non-linear radial basis function kernel [214], can recognize a potentially wider range or novel malware offline, while requiring sophisticated analysis and complex hardware implementation. With the conjecture that it is possible to build an online malware detector [51], a lightweight online hardware-supported malware detector [176] takes more types of micro-architectural features as inputs to two supervised models (i.e., logistic regression and MLPs), displaying excellent performance. To leverage software features distilled from both static and dynamic analysis, Yuan *et al.* [253] adopt the deep belief network in a semi-supervised training procedure that is composed of unsupervised pre-training and supervised fine-tuning, to classify malware in Android applications.

4.7 Code Generation and Compiler

4.7.1 Code Generation. Due to the similarities in semantics and syntax between programming languages and natural languages, the problem of code generation or translation is often modeled as an NLP problem of predicting sentences' probabilities or neural machine translation (NMT), respectively.

Targeting code completion, Raychev *et al.* [188] explore several statistical language models (i.e., the N-gram model, RNN, and a combination of these two) that extract sequences from partial programs with holes to predict potential candidates, where sentences with the highest probability and satisfying constraints of each hole are selected. As for code generation, CLgen [42] employs LSTMs to model semantics and structure of OpenCL programs from a huge corpus of hand-written codes, and generates human-like programs via iteratively sampling from the learned model.

Targeting program translation, NMT-based techniques are widely applied to migrate codes from one language to another. For example, a tree-to-tree model with the encoder-decoder structure [29] effectively translates programs from source trees to target trees; the sequence-to-sequence (seq2seq) model can also be used to translate CUDA to OpenCL [118]. Rather than translating between high-level programming languages, Coda [66] translates binary executables to the corresponding high-level code, and decomposes the decompilation into two phases: code sketch generation that employs instruction-type-aware encoder and a tree decoder with attention feeding, and iterative error correction based on an ensembled RNN-based error predictor. NMT-based techniques are also applicable to cross-architecture code similarity comparison. Zuo *et al.* [265] propose an LSTM-based cross-(assembly)-lingual basic-block embedding model. This model converts a basic block into an embedding, so that the similarity of two basic blocks can be detected by measuring the distance between their embeddings. It is noteworthy that these supervised NMT-based techniques may confront several issues: difficulty to generalize to programs longer than training ones, limited sizes of vocabulary sets, and the scarcity of aligned input-output data. Fully relying on unsupervised machine translation, TransCoder [127] can exclusively use monolingual source codes and easily generalized to other programming languages.

4.7.2 Compiler. The complexity of compilers grows with the complexity of computer architectures and workloads. ML-based techniques can optimize different perspectives of compilers, such as instruction scheduling, compiler heuristics, the order to apply optimizations, hot path identification, auto-vectorization, and compilation for specific applications.

For instruction scheduling, the temporal difference algorithm in RL can be leveraged to compute the preference function of one scheduling over another [157], which is further improved by combined with a rollout approach [156]. Regarding scheduling under highly-constrained code optimization, projective reparameterization [93], which differentiably constrains the output of NNs onto convex sets of feasible solutions, enables automatic instruction scheduling under constrains of data-dependent partial orders over the instructions.

For improving compiler heuristics, Coons *et al.* [39] employ an RL technique, Neuro-Evolution of Augmenting Topologies (NEAT), to improve the instruction placement heuristic by tuning placement cost functions. To get rid of manual feature engineering, a DNN model is developed to learn compiler heuristics from raw codes automatically [41]. It uses an LSTM-based model to extract semantics and syntactic patterns of programs, followed by a dense NN to build heuristics, so as to construct proper embeddings of program codes and simultaneously learn the optimization process.

For choosing the appropriate order to apply different optimizations, Agakov *et al.* [3] develop models (an independent distribution model and a Markov model) to predict regions of the optimization space that are more likely to bring great performance, which effectively shrinks the search space to speedup the iterative compilation. To directly find good orderings, NEAT [121] can automatically generate beneficial optimization orderings for each method in a program.

For path profiling, CrystalBall [254] uses an LSTM model to statically identify hot paths, sequences of instructions that are frequently executed. As CrystalBall only relies on intermediate representation, it avoids manual feature crafting and is independent of language or platform.

For automatic vectorization, which is crucial to enhance performance of compute-intensive programs on modern processors equipped with single instruction multiple data (SIMD), it allows compilers to exploit better data-level parallelism. Auto-vectorization means automatic conversion from scalar code to vector code, which is adopted in most production compilers, such as Intel's ICC, GNU GCC, PGI's pgcc, IBM's XL/C, etc. Mendis *et al.* [159] leverage imitation learning to train an agent modeled by a gated GNN, whose policy aims to mimic optimal vectorization decisions.

For compilation of specific applications, there are studies improving compilation for approximate computing or DNN applications. Considering compilation for approximate computing, a program transformation [60] is proposed, which trains MLPs to mimic the regions of approximable imperative code and eventually replace the original codes with trained MLPs. The following work [248] extends this algorithmic transformation to GPUs. Considering compilation for DNNs, RELEASE [5] utilizes RL to search optimal compilation configurations for DNNs, which integrates an adaptive sampling algorithm that can reduce the number of samples required to navigate the search space.

4.8 Chip Design

As the technology scales down, the increased design complexity comes with the growing process variations and reduced design margin, making chip design and manufacturing an overwhelmingly complex problem for human designers. Recent advancements in ML create a chance to transform chip design workflows.

From the analog circuit level, GCN-RL circuit designer [229] combines RL with graph convolutional networks (GCNs) for automatic transistor sizing, leveraging the analogy that circuits can be converted into graphs with vertices as transistors and edges as wires; with graph embeddings of domain knowledge, it is able to generalize across different circuit topologies or different technology nodes, AutoCkt [199] also uses deep RL, which aims to find post-layout circuit parameters to satisfy a target design specification. AutoCkt is trained on a sparse sub-sample of the design space, which improves the convergence speed and achieves 40× speedup over a traditional genetic algorithm.

From the chip level, chip placement optimization is a popular topic. Aiming at flip-flop placement optimization in clock networks, this problem can be disentangled as a post-placement flip-flop clustering by a modified K-means clustering, and the relocation of these clusters [241]. The goal is to reduce the wirelength of clock networks by reducing the distance between flip-flops and their drivers, while minimizing the disruption of original placement results. Aiming at chip placement, Mirhoseini *et al.* [161] use deep RL to place macros (memory cells), after which standard cells are placed by a force-directed method. A supervised GCN is used to encode topological information of chip netlists, generate graph embeddings as inputs to the RL agent, and provide proxy rewards to guide the search. This method is able to generalize to unseen netlists, and outperforms RePlAce [34] yet several times slower. DREAMPlace [143] focuses on placing standard cells in very-large-scale integrated (VLSI) circuits, where the classical analytical placement optimization is cast into a neural network training problem, achieving over 30× speedup without quality degradation compared to RePlAce [34]. Moreover, ML-based techniques can be applied in different steps of chip design flow, including pre-silicon hotspot detection by classification-based models (e.g., ANNs or SVMs), post-silicon variation extraction by sparse Bayesian learning, and post-silicon timing tuning to mitigate the effects caused by process variation [264].

5 DISCUSSION AND POTENTIAL DIRECTIONS

5.1 Bridging Data Gaps

Data are the backbone to ML, however, sometimes the perfect datasets are non-available or intolerably expensive, and there is no standardized dataset in the computer architecture and system domain. Here, we would like to shed light on two points, the gap between small data and big data, and non-perfect data.

In some EDA problems, such as chip placement, the simulation or evaluation is extremely expensive, resulting in data scarcity. As ML models usually require enough data to learn the statistics and make decisions, this gap between small data and big data often limits the capability of ML-based techniques. There have been different attempts to bridge this gap: from the algorithm side, algorithms that can work with small data await to be developed, where one current technique is Bayesian optimization that is effective when the parameter space is small [115]; from the data side, generative methods can be used to generate synthetic data [53], mitigating data scarcity.

In terms of non-perfect data, even though some EDA tools produce a lot of data, they are not always labeled nor properly presented in the form suitable to ML. In the absence of perfectly labeled training data, possible alternatives are to use unsupervised learning, self-supervised learning [82], or to combine supervised with unsupervised techniques [6]. Meanwhile, RL can also be used, which can generate training data on the fly via trial and error.

5.2 Developing Algorithms

Although there have been a lot of accomplishments, we are still expecting novel ML algorithms or schemes to further improve both modelling and system optimization. With increasingly growing system complexity, these algorithms should be scalable such that the running overhead is always tolerable. ML-based techniques are often considered as black-box optimization, but sometimes we do need clear model interpretabiliity and assistance from domain knowledge.

New ML schemes. Existing studies generally apply ML based on single-level abstractions. As classical analytic-based methods work in either bottom-up or top-down manners, these limitations of ML-based design encourage developments of algorithms to distill hierarchical structures of systems/architecture. One example is hierarchical RL [122], which has flexible goal specifications and is talented to learn goal-directed behaviors in complex environments with sparse feedback. Such kind of models enables more flexible and effective multi-level design and control. Additionally, many system optimizations involve participation of multiple agents, such as NoC routing, which are naturally suitable to the realm of multi-agent RL (MARL) [257]. These agents can be fully cooperative, fully competitive, or a mix of the two, enabling versatility of system optimization.

Another promising approach is self-supervised learning [82], beneficial in both improving model robustness and mitigating data scarcity.

While applying a single ML method solely has led to powerful results, hybrid methods, i.e., combining different ML techniques or combining ML techniques with heuristics, unleash more opportunities. For example, supervised learning can cooperate with unsupervised learning for malware detection [253]; RL can be combined with genetic algorithms for hardware resource assignment [114].

Scalability. The system scaling-up poses challenges on the scalability issues. From the algorithm side, multi-level techniques can help reduce the computation complexity, e.g., multi-level Q-learning for DVFS [32, 33, 177]. One implicit workaround is to leverage transfer learning: the pre-training is a one-time cost, which can be amortized in each future use; the fine-tuning provides flexibility between a quick solution from the pre-trained model and a longer yet better one for a particular task. Several examples [161, 199, 229] are discussed in Section 4.8.

Domain Knowledge and Interpretability. Not only can domain knowledge improve the interpretability of ML models, it could also be helpful in model/algorithm selection and optimization. Making better use of domain knowledge unveils possibilities to choose more proper models dealing with different system problems and provide more intuitions or explanations of why and how these models work. By making analogy of semantics/syntax between memory access patterns or program language and natural languages, these prefetching or code generation problems can be modeled as NLP problems, as discussed in Section 4.1.1 and Section 4.7.1. By making analogy of graphical representations in many EDA problems, where data are intrinsically presented as graphs (e.g., circuits, logic netlists or intermediate representations), GNNs are expected to be powerful in these fields [115]. Several examples are provided in Section 4.8.

5.3 Improving Implementations and Deployments

To fully exploit advantages of ML-based methods, we need efficient strategies for practical implementation with reasonable overheads, and we also need to carefully consider deployment scenarios.

Better implementations. To enable practical implementations of ML-based techniques, improvement can be made from either the model side or the software/hardware co-design [212]. From the model level, network pruning and model compression reduce the number of operations and model size [79]; weight quantization improves computation efficiency by reducing the precision of operations/operands [92]. From the co-design level, strategies that have been used for DNN acceleration could also be used in applying ML for system.

Appropriate scenarios: online vs. offline. When deploying ML-based techniques for system designs, it is crucial to deliberate the design constraints under different scenarios. Generally, existing work falls into two categories. The first one is to apply ML-based techniques online or during runtime, no matter the training phase is performed online or offline. Obviously, the model complexity and runtime overhead are often strictly limited by specific constraints, e.g., power/energy, timing/latency, area, etc. To take one more step, if the online training/learning is further desired, the design constraint will be more stringent. One promising approach is to employ semi-online learning models, which have been applied to solve some classical combinatorial optimization problems, such as bipartite matching [123] and caching [124]. These models enable smooth interpolation between the best possible online and offline training algorithms. The second one is to apply ML-based techniques offline, which usually refers to architectural design space exploration. Such problems leverage ML-based techniques to guide system implementation, and once the designing phase is completed, the ML models will not be invoked again. In consequence, these offline applications can adopt more complex ML techniques that may bring higher overheads.

5.4 Supporting Novel Applications

ML-based techniques are supposed to be applicable in both currently existing architectures and emerging systems, leading to long-term evolvement and advancement in computer architecture and systems. Notably, some design areas are evergreen and some issues are universal in system design. Several examples include caching in hardware/software/data centers (Section 4.1.1 and Section 4.5), scheduling in multi-core CPUs and PIM-assisted GPU architectures (Section 4.4.3), resource management and task allocation in single/multi/many-core CPUs and heterogeneous systems (Section 4.4), NoC design under various scenarios (Section 4.3), etc. Even with limited knowledge of novel system problems, transfer learning and meta-learning [173, 222] could also be beneficial in either exploring new and better heuristics or directly deriving design methodology, guaranteeing reliable guidance and strong performance in system design.

5.5 Designing General Tools

One ultimate goal of applying ML for computer architecture and system might be the fully automatic design, which should entangle two principal capabilities: the holistic optimization in system-wise under multiple objectives, the easiness to immigrate across different systems so as to enable rapid and agile design.

Holistic optimization. Fueled by advancements in ML, there are explorations to broader ML-based system design and optimization strategies [50]. They could be multi-objective optimizations, or optimizing several components in a system simultaneously. We further envisage an ML-based system-wise, holistic framework that has a panoramic vision and conducts optimization during run-time: it should be able to take advantage of information/features from different levels of systems in synergy, so that it could thoroughly characterize and learn system behaviors as well as their intrinsically hierarchical abstractions; it should also be able to make decisions in different granularity, so that it could control and improve systems precisely and comprehensively.

Portable, rapid, and agile. Striving for portable, rapid, and agile design, there are two potential directions. The first one is to carefully design the interface between systems/architectures and ML-based techniques. As ML-based techniques can perform well without accurate and explicit description of domains, they could open up the portability across different systems. The other one is endeavor to build ML-based design automation tools. ML-based techniques have more or less transformed the workflow of design automation, from either modelling or automated exploration perspective [115]. We expect GNNs make better use of the naturally graphical data in EDA field; we expect deep RL be a powerful and general-purpose tool for many EDA optimization problems, especially when the exact heuristic or objective is obscure; we expect these ML-based design automation tools enhance designers' productivity and thrive in the community.

6 CONCLUSION

The flourishing of ML would be retarded without the great systems and powerful architectures supportive to run these algorithms at scale. Now, it is the time to return the favor and let ML transform the way that computer architecture and systems are designed. Existing work that applies ML for system roughly falls into two categories: ML-based modelling that involves performance metrics or some other criteria of interest, and ML-based design methodology that directly leverages ML as the design tool. We further present a future vision of opportunities and potential directions, which may bring a brighter and more promising future of applying ML for computer architecture and systems. We hope to see the virtuous cycle, in which ML-based techniques are efficiently running on the most powerful computers with the pursuit of designing the next generation computers. We hope ML-based techniques could be the impetus to the revolution of computer architecture and systems.

REFERENCES

[1] Nevine AbouGhazaleh, Alexandre Ferreira, Cosmin Rusu, Ruibin Xu, Frank Liberato, Bruce Childers, Daniel Mosse, and Rami Melhem. 2007. Integrated CPU and L2 cache voltage scaling using machine learning. In *ACM SIGPLAN Notices*, Vol. 42. ACM, 41–50.

[2] Ravichandra Addanki, Shaileshh Bojja Venkatakrishnan, Shreyan Gupta, Hongzi Mao, and Mohammad Alizadeh. 2018. Placeto: Efficient Progressive Device Placement Optimization. In *NIPS Machine Learning for Systems Workshop*.

[3] Felix Agakov, Edwin Bonilla, John Cavazos, Björn Franke, Grigori Fursin, Michael FP O'Boyle, John Thomson, Marc Toussaint, and Christopher KI Williams. 2006. Using machine learning to focus iterative optimization. In *Proceedings of the international symposium on code generation and optimization*. IEEE Computer Society, 295–305.

[4] Nitish Agarwal, Tulsi Jain, and Mohamed Zahran. 2019. Performance Prediction for Multi-threaded Applications. In *International Workshop on AI-assisted Design for Architecture (AIDArc), held in conjunction with ISCA*.

[5] Byung Hoon Ahn, Prannoy Pilligundla, and Hadi Esmaeilzadeh. 2019. Reinforcement Learning and Adaptive Sampling for Optimized DNN Compilation. *arXiv preprint arXiv:1905.12799* (2019).

[6] Mohamad Alawieh, Fa Wang, and Xin Li. 2017. Efficient hierarchical performance modeling for integrated circuits via bayesian co-learning. In *Proceedings of the 54th Annual Design Automation Conference 2017*. ACM, 9.

[7] Aayush Ankit, Abhronil Sengupta, Priyadarshini Panda, and Kaushik Roy. 2017. Resparc: A reconfigurable and energy-efficient architecture with memristive crossbars for deep spiking neural networks. In *Proceedings of the 54th Annual Design Automation Conference 2017*. ACM, 27.

[8] Newsha Ardalani, Clint Lestourgeon, Karthikeyan Sankaralingam, and Xiaojin Zhu. 2015. Cross-architecture performance prediction (XAPP) using CPU code to predict GPU performance. In *Proceedings of the 48th International Symposium on Microarchitecture*. ACM, 725–737.

[9] Newsha Ardalani, Urmish Thakker, Aws Albarghouthi, and Karu Sankaralingam. 2019. A Static Analysis-based Cross-Architecture Performance Prediction Using Machine Learning. *arXiv preprint arXiv:1906.07840* (2019).

[10] Yuxin Bai, Victor W Lee, and Engin Ipek. 2017. Voltage regulator efficiency aware power management. *ACM SIGOPS Operating Systems Review* 51, 2 (2017), 825–838.

[11] Peter E Bailey, David K Lowenthal, Vignesh Ravi, Barry Rountree, Martin Schulz, and Bronis R De Supinski. 2014. Adaptive configuration selection for power-constrained heterogeneous systems. In *2014 43rd International Conference on Parallel Processing*. IEEE, 371–380.

[12] Rajeev Balasubramonian, Jichuan Chang, Troy Manning, Jaime H Moreno, Richard Murphy, Ravi Nair, and Steven Swanson. 2014. Near-data processing: Insights from a MICRO-46 workshop. *IEEE Micro* 34, 4 (2014), 36–42.

[13] Ioana Baldini, Stephen J Fink, and Erik Altman. 2014. Predicting gpu performance from cpu runs using machine learning. In *2014 IEEE 26th International Symposium on Computer Architecture and High Performance Computing*. IEEE, 254–261.

[14] Nathan Beckmann and Daniel Sanchez. 2017. Maximizing cache performance under uncertainty. In *2017 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 109–120.

[15] Christopher Berner, Greg Brockman, Brooke Chan, Vicki Cheung, Przemysław Dębiak, Christy Dennison, David Farhi, Quirin Fischer, Shariq Hashme, Chris Hesse, et al. 2019. Dota 2 with Large Scale Deep Reinforcement Learning. *arXiv preprint arXiv:1912.06680* (2019).

[16] Eshan Bhatia, Gino Chacon, Seth Pugsley, Elvira Teran, Paul V Gratz, and Daniel A Jiménez. 2019. Perceptron-based prefetch filtering. In *Proceedings of the 46th International Symposium on Computer Architecture*. ACM, 1–13.

[17] Andrea Di Biagio. 2018. llvm-mca: a static performance analysis tool. (2018). https://lists.llvm.org/pipermail/llvm-dev/2018-March/121490.html

[18] Nathan Binkert, Bradford Beckmann, Gabriel Black, Steven K Reinhardt, Ali Saidi, Arkaprava Basu, Joel Hestness, Derek R Hower, Tushar Krishna, Somayeh Sardashti, et al. 2011. The gem5 simulator. *ACM SIGARCH Computer Architecture News* 39, 2 (2011), 1–7.

[19] Ramazan Bitirgen, Engin Ipek, and Jose F Martinez. 2008. Coordinated management of multiple interacting resources in chip multiprocessors: A machine learning approach. In *2008 41st IEEE/ACM International Symposium on Microarchitecture*. IEEE, 318–329.

[20] Ronald D Blanton, Xin Li, Ken Mai, Diana Marculescu, Radu Marculescu, Jeyanandh Paramesh, Jeff Schneider, and Donald E Thomas. 2015. Statistical learning in chip (SLIC). In *2015 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. IEEE, 664–669.

[21] Shekhar Borkar. 2013. Exascale computing–a fact or affliction. *Keynote presentation at IPDPS* 10 (2013).

[22] Justin Boyan and Andrew W Moore. 2000. Learning evaluation functions to improve optimization by local search. *Journal of Machine Learning Research* 1, Nov (2000), 77–112.

[23] Justin A Boyan and Michael L Littman. 1994. Packet routing in dynamically changing networks: A reinforcement learning approach. In *Advances in neural information processing systems*. 671–678.

[24] Peter Braun and Heiner Litz. 2019. Understanding Memory Access Patterns for Prefetching. In *International Workshop on AI-assisted Design for Architecture (AIDArc), held in conjunction with ISCA*.

[25] Geoffrey W Burr, Matthew J Brightsky, Abu Sebastian, Huai-Yu Cheng, Jau-Yi Wu, Sangbum Kim, Norma E Sosa, Nikolaos Papandreou, Hsiang-Lan Lung, Haralampos Pozidis, et al. 2016. Recent progress in phase-change memory technology. *IEEE Journal on Emerging and Selected Topics in Circuits and Systems* 6, 2 (2016), 146–162.

[26] Brad Calder, Dirk Grunwald, Michael Jones, Donald Lindsay, James Martin, Michael Mozer, and Benjamin Zorn. 1997. Evidence-based static branch prediction using machine learning. *ACM Transactions on Programming Languages and Systems (TOPLAS)* 19, 1 (1997), 188–222.

[27] Rodrigo N Calheiros, Enayat Masoumi, Rajiv Ranjan, and Rajkumar Buyya. 2014. Workload prediction using ARIMA model and its impact on cloud applications' QoS. *IEEE Transactions on Cloud Computing* 3, 4 (2014), 449–458.

[28] Li Chen, Justinas Lingys, Kai Chen, and Feng Liu. 2018. Auto: Scaling deep reinforcement learning for datacenter-scale automatic traffic optimization. In *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication*. 191–205.

[29] Xinyun Chen, Chang Liu, and Dawn Song. 2018. Tree-to-tree neural networks for program translation. *Advances in neural information processing systems* 31 (2018), 2547–2557.

[30] Yunji Chen, Tao Luo, Shaoli Liu, Shijin Zhang, Liqiang He, Jia Wang, Ling Li, Tianshi Chen, Zhiwei Xu, Ninghui Sun, et al. 2014. Dadiannao: A machine-learning supercomputer. In *Proceedings of the 47th Annual IEEE/ACM International Symposium on Microarchitecture*. IEEE Computer Society, 609–622.

[31] Yu-Hsin Chen, Joel Emer, and Vivienne Sze. 2016. Eyeriss: A spatial architecture for energy-efficient dataflow for convolutional neural networks. In *ACM SIGARCH Computer Architecture News*, Vol. 44. IEEE Press, 367–379.

[32] Zhuo Chen and Diana Marculescu. 2015. Distributed reinforcement learning for power limited many-core system performance optimization. In *Proceedings of the 2015 Design, Automation & Test in Europe Conference & Exhibition*. EDA Consortium, 1521–1526.

[33] Zhuo Chen, Dimitrios Stamoulis, and Diana Marculescu. 2017. Profit: priority and power/performance optimization for many-core systems. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 37, 10 (2017), 2064–2075.

[34] Chung-Kuan Cheng, Andrew B Kahng, Ilgweon Kang, and Lutong Wang. 2018. Replace: Advancing solution quality and routability validation in global placement. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 38, 9 (2018), 1717–1730.

[35] Ping Chi, Shuangchen Li, Cong Xu, Tao Zhang, Jishen Zhao, Yongpan Liu, Yu Wang, and Yuan Xie. 2016. Prime: A novel processing-in-memory architecture for neural network computation in reram-based main memory. In *ACM SIGARCH Computer Architecture News*, Vol. 44. IEEE Press, 27–39.

[36] Mark Clark, Avinash Kodi, Razvan Bunescu, and Ahmed Louri. 2018. LEAD: Learning-enabled energy-aware dynamic voltage/frequency scaling in NoCs. In *Proceedings of the 55th Annual Design Automation Conference*. ACM, 82.

[37] Ryan Cochran, Can Hankendi, Ayse K Coskun, and Sherief Reda. 2011. Pack & Cap: adaptive DVFS and thread packing under power caps. In *2011 44th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 175–185.

[38] Ronan Collobert, Jason Weston, Léon Bottou, Michael Karlen, Koray Kavukcuoglu, and Pavel Kuksa. 2011. Natural language processing (almost) from scratch. *Journal of machine learning research* 12, Aug (2011), 2493–2537.

[39] Katherine E Coons, Behnam Robatmili, Matthew E Taylor, Bertrand A Maher, Doug Burger, and Kathryn S McKinley. 2008. Feature selection and policy optimization for distributed instruction placement using reinforcement learning. In *Proceedings of the 17th international conference on Parallel architectures and compilation techniques*. ACM, 32–42.

[40] Eli Cortez, Anand Bonde, Alexandre Muzio, Mark Russinovich, Marcus Fontoura, and Ricardo Bianchini. 2017. Resource central: Understanding and predicting workloads for improved resource management in large cloud platforms. In *Proceedings of the 26th Symposium on Operating Systems Principles*. 153–167.

[41] Chris Cummins, Pavlos Petoumenos, Zheng Wang, and Hugh Leather. 2017. End-to-end deep learning of optimization heuristics. In *2017 26th International Conference on Parallel Architectures and Compilation Techniques (PACT)*. IEEE, 219–232.

[42] Chris Cummins, Pavlos Petoumenos, Zheng Wang, and Hugh Leather. 2017. Synthesizing benchmarks for predictive modeling. In *2017 IEEE/ACM International Symposium on Code Generation and Optimization (CGO)*. IEEE, 86–99.

[43] Matthew Curtis-Maury, James Dzierwa, Christos D Antonopoulos, and Dimitrios S Nikolopoulos. 2006. Online power-performance adaptation of multithreaded programs using hardware event-based prediction. In *Proceedings of the 20th annual international conference on Supercomputing*. 157–166.

[44] Matthew Curtis-Maury, Ankur Shah, Filip Blagojevic, Dimitrios S Nikolopoulos, Bronis R De Supinski, and Martin Schulz. 2008. Prediction models for multi-dimensional power-performance optimization on many cores. In *Proceedings of the 17th international conference on Parallel architectures and compilation techniques*. 250–259.

[45] Matthew Curtis-Maury, Karan Singh, Sally A McKee, Filip Blagojevic, Dimitrios S Nikolopoulos, Bronis R De Supinski, and Martin Schulz. 2007. Identifying energy-efficient concurrency levels using machine learning. In *2007 IEEE International Conference on Cluster Computing*. IEEE, 488–495.

[46] Dong Dai, Forrest Sheng Bao, Jiang Zhou, and Yong Chen. 2016. Block2vec: A deep learning strategy on mining block correlations in storage systems. In *2016 45th International Conference on Parallel Processing Workshops (ICPPW)*. IEEE, 230–239.

[47] Sourav Das, Janardhan Rao Doppa, Dae Hyun Kim, Partha Pratim Pande, and Krishnendu Chakrabarty. 2015. Optimizing 3D NoC design for energy efficiency: A machine learning approach. In *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design*. IEEE Press, 705–712.

[48] Sourav Das, Janardhan Rao Doppa, Partha Pratim Pande, and Krishnendu Chakrabarty. 2016. Energy-efficient and reliable 3D Network-on-Chip (NoC): Architectures and optimization algorithms. In *Proceedings of the 35th International Conference on Computer-Aided Design*. ACM, 57.

[49] Bhavya K Daya, Li-Shiuan Peh, and Anantha P Chandrakasan. 2016. Quest for high-performance bufferless NoCs with single-cycle express paths and self-learning throttling. In *2016 53nd ACM/EDAC/IEEE Design Automation Conference (DAC)*. IEEE, 1–6.

[50] Jeffrey Dean. 2020. 1.1 the deep learning revolution and its implications for computer architecture and chip design. In *2020 IEEE International Solid-State Circuits Conference-(ISSCC)*. IEEE, 8–14.

[51] John Demme, Matthew Maycock, Jared Schmitz, Adrian Tang, Adam Waksman, Simha Sethumadhavan, and Salvatore Stolfo. 2013. On the feasibility of online malware detection with performance counters. *ACM SIGARCH Computer Architecture News* 41, 3 (2013), 559–570.

[52] Zhaoxia Deng, Lunkai Zhang, Nikita Mishra, Henry Hoffmann, and Frederic T Chong. 2017. Memory cocktail therapy: a general learning-based framework to optimize dynamic tradeoffs in NVMs. In *Proceedings of the 50th Annual IEEE/ACM International Symposium on Microarchitecture*. ACM, 232–244.

[53] Yi Ding, Nikita Mishra, and Henry Hoffmann. 2019. Generative and multi-phase learning for computer systems optimization. In *Proceedings of the 46th International Symposium on Computer Architecture*. ACM, 39–52.

[54] Dominic DiTomaso, Travis Boraten, Avinash Kodi, and Ahmed Louri. 2016. Dynamic error mitigation in NoCs using intelligent prediction techniques. In *The 49th Annual IEEE/ACM International Symposium on Microarchitecture*. IEEE Press, 31.

[55] Dominic DiTomaso, Ashif Sikder, Avinash Kodi, and Ahmed Louri. 2017. Machine learning enabled power-aware network-on-chip design. In *Proceedings of the Conference on Design, Automation & Test in Europe*. European Design and Automation Association, 1354–1359.

[56] Xiangyu Dong, Norman P Jouppi, and Yuan Xie. 2013. A circuit-architecture co-optimization framework for exploring nonvolatile memory hierarchies. *ACM Transactions on Architecture and Code Optimization (TACO)* 10, 4 (2013), 23.

[57] Zidong Du, Robert Fasthuber, Tianshi Chen, Paolo Ienne, Ling Li, Tao Luo, Xiaobing Feng, Yunji Chen, and Olivier Temam. 2015. ShiDianNao: Shifting vision processing closer to the sensor. In *ACM SIGARCH Computer Architecture News*, Vol. 43. ACM, 92–104.

[58] Christophe Dubach, Timothy M Jones, Edwin V Bonilla, and Michael FP O'Boyle. 2010. A predictive model for dynamic microarchitectural adaptivity control. In *Proceedings of the 2010 43rd Annual IEEE/ACM International Symposium on Microarchitecture*. IEEE Computer Society, 485–496.

[59] Masoumeh Ebrahimi, Masoud Daneshtalab, Fahimeh Farahnakian, Juha Plosila, Pasi Liljeberg, Maurizio Palesi, and Hannu Tenhunen. 2012. HARAQ: Congestion-aware learning model for highly adaptive routing algorithm in on-chip networks. In *2012 IEEE/ACM Sixth International Symposium on Networks-on-Chip*. IEEE, 19–26.

[60] Hadi Esmaeilzadeh, Adrian Sampson, Luis Ceze, and Doug Burger. 2012. Neural acceleration for general-purpose approximate programs. In *Proceedings of the 2012 45th Annual IEEE/ACM International Symposium on Microarchitecture*. IEEE Computer Society, 449–460.

[61] Sérgio Esteves, Helena Galhardas, and Luís Veiga. 2018. Adaptive Execution of Continuous and Data-intensive Workflows with Machine Learning. In *Proceedings of the 19th International Middleware Conference*. 239–252.

[62] Stijn Eyerman, Kenneth Hoste, and Lieven Eeckhout. 2011. Mechanistic-empirical processor performance modeling for constructing CPI stacks on real hardware. In *(IEEE ISPASS) IEEE International Symposium on Performance Analysis of Systems and Software*. IEEE, 216–226.

[63] Alexandra Fedorova, David Vengerov, and Daniel Doucette. 2007. Operating system scheduling on heterogeneous core systems. In *Proceedings of the Workshop on Operating System Support for Heterogeneous Multicore Architectures*.

[64] Chaochao Feng, Zhonghai Lu, Axel Jantsch, Jinwen Li, and Minxuan Zhang. 2010. A reconfigurable fault-tolerant deflection routing algorithm based on reinforcement learning for network-on-chip. In *Proceedings of the Third*

International Workshop on Network on Chip Architectures. ACM, 11–16.
[65] Quintin Fettes, Mark Clark, Razvan Bunescu, Avinash Karanth, and Ahmed Louri. 2019. Dynamic Voltage and Frequency Scaling in NoCs with Supervised and Reinforcement Learning Techniques. *IEEE Trans. Comput.* 68, 3 (2019).
[66] Cheng Fu, Huili Chen, Haolan Liu, Xinyun Chen, Yuandong Tian, Farinaz Koushanfar, and Jishen Zhao. 2019. Coda: An End-to-End Neural Program Decompiler. In *Advances in Neural Information Processing Systems*. 3703–3714.
[67] Archana Ganapathi, Kaushik Datta, Armando Fox, and David Patterson. 2009. A case for machine learning to optimize multicore performance. In *Proceedings of the First USENIX conference on Hot topics in parallelism*. USENIX Association Berkeley, CA, 1–1.
[68] Archana Ganapathi, Harumi Kuno, Umeshwar Dayal, Janet L Wiener, Armando Fox, Michael Jordan, and David Patterson. 2009. Predicting multiple metrics for queries: Better decisions enabled by machine learning. In *2009 IEEE 25th International Conference on Data Engineering*. IEEE, 592–603.
[69] Jim Gao. 2014. Machine learning applications for data center optimization. (2014).
[70] Yuanxiang Gao, Li Chen, and Baochun Li. 2018. Post: Device placement with cross-entropy minimization and proximal policy optimization. In *Advances in Neural Information Processing Systems*. 9971–9980.
[71] Yuanxiang Gao, Li Chen, and Baochun Li. 2018. Spotlight: Optimizing device placement for training deep neural networks. In *International Conference on Machine Learning*. 1662–1670.
[72] Elba Garza, Samira Mirbagher-Ajorpaz, Tahsin Ahmad Khan, and Daniel A Jiménez. 2019. Bit-level perceptron prediction for indirect branches. In *Proceedings of the 46th International Symposium on Computer Architecture*. ACM, 27–38.
[73] Maya Gokhale, Bill Holmes, and Ken Iobst. 1995. Processing in memory: The Terasys massively parallel PIM array. *Computer* 28, 4 (1995), 23–31.
[74] Faustino J Gomez, Doug Burger, and Risto Miikkulainen. 2001. A neuro-evolution method for dynamic resource allocation on a chip multiprocessor. In *IJCNN'01. International Joint Conference on Neural Networks. Proceedings (Cat. No. 01CH37222)*, Vol. 4. IEEE, 2355–2360.
[75] Zhenhuan Gong, Xiaohui Gu, and John Wilkes. 2010. Press: Predictive elastic resource scaling for cloud systems. In *2010 International Conference on Network and Service Management*. Ieee, 9–16.
[76] Alex Graves and Navdeep Jaitly. 2014. Towards end-to-end speech recognition with recurrent neural networks. In *International conference on machine learning*. 1764–1772.
[77] Shixiang Gu, Ethan Holly, Timothy Lillicrap, and Sergey Levine. 2017. Deep reinforcement learning for robotic manipulation with asynchronous off-policy updates. In *2017 IEEE international conference on robotics and automation (ICRA)*. IEEE, 3389–3396.
[78] Greg Hamerly, Charles Elkan, et al. 2001. Bayesian approaches to failure prediction for disk drives. In *ICML*, Vol. 1. 202–209.
[79] Song Han, Huizi Mao, and William J Dally. 2016. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. In *International Conference on Learning Representations*.
[80] Milad Hashemi, Kevin Swersky, Jamie Smith, Grant Ayers, Heiner Litz, Jichuan Chang, Christos Kozyrakis, and Parthasarathy Ranganathan. 2018. Learning Memory Access Patterns. In *International Conference on Machine Learning*. 1924–1933.
[81] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 770–778.
[82] Dan Hendrycks, Mantas Mazeika, Saurav Kadavath, and Dawn Song. 2019. Using self-supervised learning can improve model robustness and uncertainty. *arXiv preprint arXiv:1906.12340* (2019).
[83] Geoffrey Hinton, Li Deng, Dong Yu, George Dahl, Abdel-rahman Mohamed, Navdeep Jaitly, Andrew Senior, Vincent Vanhoucke, Patrick Nguyen, Brian Kingsbury, et al. 2012. Deep neural networks for acoustic modeling in speech recognition. *IEEE Signal processing magazine* 29 (2012).
[84] Henry Hoffmann. 2015. JouleGuard: energy guarantees for approximate applications. In *Proceedings of the 25th Symposium on Operating Systems Principles*. 198–214.
[85] M Hosomi, H Yamagishi, T Yamamoto, K Bessho, Y Higo, K Yamane, H Yamada, M Shoji, H Hachino, C Fukumoto, et al. 2005. A novel nonvolatile memory with spin torque transfer magnetization switching: Spin-RAM. In *IEEE InternationalElectron Devices Meeting, 2005. IEDM Technical Digest*. IEEE, 459–462.
[86] Jemin Hwangbo, Joonho Lee, Alexey Dosovitskiy, Dario Bellicoso, Vassilios Tsounis, Vladlen Koltun, and Marco Hutter. 2019. Learning agile and dynamic motor skills for legged robots. *Science Robotics* 4, ARTICLE (2019), eaau5872.
[87] Connor Imes, Steven Hofmeyr, and Henry Hoffmann. 2018. Energy-efficient application resource scheduling using machine learning classifiers. In *Proceedings of the 47th International Conference on Parallel Processing*. ACM, 45.
[88] Intel. 2017. Intel Architecture Code Analyzer. (2017). https://software.intel.com/en-us/articles/intel-architecture-code-analyzer

[89] Engin İpek, Sally A McKee, Rich Caruana, Bronis R de Supinski, and Martin Schulz. 2006. *Efficiently exploring architectural design spaces via predictive modeling*. Vol. 41. ACM.

[90] E Ipek, O Mutlu, JF Martinez, and R Caruana. 2008. Self-Optimizing Memory Controllers: A Reinforcement Learning Approach. In *2008 International Symposium on Computer Architecture (ISCA)*. ACM.

[91] Sadeka Islam, Jacky Keung, Kevin Lee, and Anna Liu. 2012. Empirical prediction models for adaptive resource provisioning in the cloud. *Future Generation Computer Systems* 28, 1 (2012), 155–162.

[92] Benoit Jacob, Skirmantas Kligys, Bo Chen, Menglong Zhu, Matthew Tang, Andrew Howard, Hartwig Adam, and Dmitry Kalenichenko. 2018. Quantization and training of neural networks for efficient integer-arithmetic-only inference. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2704–2713.

[93] Ajay Jain and Saman Amarasinghe. 2019. Learning automatic schedulers with projective reparameterization. In *Proceedings of the ML-for-Systems Workshop at the 46th International Symposium on Computer Architecture (ISCA'19)*.

[94] Anil K Jain. 2010. Data clustering: 50 years beyond K-means. *Pattern recognition letters* 31, 8 (2010), 651–666.

[95] Rahul Jain, Preeti Ranjan Panda, and Sreenivas Subramoney. 2016. Machine learned machines: adaptive co-optimization of caches, cores, and on-chip network. In *2016 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 253–256.

[96] Wenhao Jia, Kelly A Shaw, and Margaret Martonosi. 2012. Stargazer: Automated regression-based GPU design space exploration. In *2012 IEEE International Symposium on Performance Analysis of Systems & Software*. IEEE, 2–13.

[97] Daniel A Jiménez. 2003. Fast path-based neural branch prediction. In *Proceedings. 36th Annual IEEE/ACM International Symposium on Microarchitecture, 2003. MICRO-36*. IEEE, 243–252.

[98] Daniel A Jiménez. 2005. Piecewise linear branch prediction. In *32nd International Symposium on Computer Architecture (ISCA'05)*. IEEE, 382–393.

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

[100] Daniel A Jiménez. 2016. Multiperspective perceptron predictor. *Championship Branch Prediction (CBP-5)* (2016).

[101] Daniel A Jiménez and Calvin Lin. 2001. Dynamic branch prediction with perceptrons. In *Proceedings HPCA Seventh International Symposium on High-Performance Computer Architecture*. IEEE, 197–206.

[102] Daniel A Jiménez and Elvira Teran. 2017. Multiperspective reuse prediction. In *2017 50th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 436–448.

[103] Wengong Jin, Connor Coley, Regina Barzilay, and Tommi Jaakkola. 2017. Predicting organic reaction outcomes with Weisfeiler-Lehman network. In *Advances in Neural Information Processing Systems*. 2607–2616.

[104] Biresh Kumar Joardar, Ryan Gary Kim, Janardhan Rao Doppa, Partha Pratim Pande, Diana Marculescu, and Radu Marculescu. 2018. Learning-based Application-Agnostic 3D NoC Design for Heterogeneous Manycore Systems. *IEEE Trans. Comput.* 68, 6 (2018), 852–866.

[105] Ali Jooya, Nikitas Dimopoulos, and Amirali Baniasadi. 2016. MultiObjective GPU design space exploration optimization. In *2016 International Conference on High Performance Computing & Simulation (HPCS)*. IEEE, 659–666.

[106] PJ Joseph, Kapil Vaswani, and Matthew J Thazhuthaveetil. 2006. Construction and use of linear regression models for processor performance analysis. In *The Twelfth International Symposium on High-Performance Computer Architecture, 2006*. IEEE, 99–108.

[107] PJ Joseph, Kapil Vaswani, and Matthew J Thazhuthaveetil. 2006. A predictive performance model for superscalar processors. In *2006 39th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO'06)*. IEEE, 161–170.

[108] Norman P Jouppi, Cliff Young, Nishant Patil, David Patterson, Gaurav Agrawal, Raminder Bajwa, Sarah Bates, Suresh Bhatia, Nan Boden, Al Borchers, et al. 2017. In-datacenter performance analysis of a tensor processing unit. In *2017 ACM/IEEE 44th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 1–12.

[109] Da-Cheng Juan, Siddharth Garg, Jinpyo Park, and Diana Marculescu. 2013. Learning the optimal operating point for many-core systems with extended range voltage/frequency scaling. In *Proceedings of the Ninth IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis*. IEEE Press, 8.

[110] Da-Cheng Juan and Diana Marculescu. 2012. 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*. 97–102.

[111] Elena Kakoulli, Vassos Soteriou, and Theocharis Theocharides. 2012. Intelligent hotspot prediction for network-on-chip-based multicore systems. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 31, 3 (2012), 418–431.

[112] Wonkyung Kang, Dongkun Shin, and Sungjoo Yoo. 2017. Reinforcement learning-assisted garbage collection to mitigate long-tail latency in SSD. *ACM Transactions on Embedded Computing Systems (TECS)* 16, 5s (2017), 1–20.

[113] Wonkyung Kang and Sungjoo Yoo. 2018. Dynamic management of key states for reinforcement learning-assisted garbage collection to reduce long tail latency in SSD. In *2018 55th ACM/ESDA/IEEE Design Automation Conference (DAC)*. IEEE, 1–6.

[114] Sheng-Chun Kao, Geonhwa Jeong, and Tushar Krishna. 2020. ConfuciuX: Autonomous Hardware Resource Assignment for DNN Accelerators using Reinforcement Learning. In *2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 622–636.

[115] Brucek Khailany, Haoxing Ren, Steve Dai, Saad Godil, Ben Keller, Robert Kirby, Alicia Klinefelter, Rangharajan Venkatesan, Yanqing Zhang, Bryan Catanzaro, et al. 2020. Accelerating Chip Design with Machine Learning. *IEEE Micro* 40, 6 (2020), 23–32.

[116] Arijit Khan, Xifeng Yan, Shu Tao, and Nikos Anerousis. 2012. Workload characterization and prediction in the cloud: A multiple time series approach. In *2012 IEEE Network Operations and Management Symposium*. IEEE, 1287–1294.

[117] Salman Khan, Polychronis Xekalakis, John Cavazos, and Marcelo Cintra. 2007. Using predictivemodeling for cross-program design space exploration in multicore systems. In *16th International Conference on Parallel Architecture and Compilation Techniques (PACT 2007)*. IEEE, 327–338.

[118] Yonghae Kim and Hyesoon Kim. 2019. A Case Study: Exploiting Neural Machine Translation to Translate CUDA to OpenCL. *arXiv preprint arXiv:1905.07653* (2019).

[119] Sotiris B Kotsiantis, I Zaharakis, and P Pintelas. 2007. Supervised machine learning: A review of classification techniques. *Emerging artificial intelligence applications in computer engineering* 160 (2007), 3–24.

[120] Tim Kraska, Alex Beutel, Ed H Chi, Jeffrey Dean, and Neoklis Polyzotis. 2018. The case for learned index structures. In *Proceedings of the 2018 International Conference on Management of Data*. ACM, 489–504.

[121] Sameer Kulkarni and John Cavazos. 2012. Mitigating the compiler optimization phase-ordering problem using machine learning. In *Proceedings of the ACM international conference on Object oriented programming systems languages and applications*. 147–162.

[122] Tejas D Kulkarni, Karthik R Narasimhan, Ardavan Saeedi, and Joshua B Tenenbaum. 2016. Hierarchical deep reinforcement learning: integrating temporal abstraction and intrinsic motivation. In *Proceedings of the 30th International Conference on Neural Information Processing Systems*. 3682–3690.

[123] Ravi Kumar, Manish Purohit, Aaron Schild, Zoya Svitkina, and Erik Vee. 2019. Semi-Online Bipartite Matching. In *Proceedings of the 10th Innovations in Theoretical Computer Science Conference, ITCS*.

[124] Ravi Kumar, Manish Purohit, Zoya Svitkina, and Erik Vee. 2020. Interleaved Caching with Access Graphs. In *Proceedings of the Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms*. SIAM, 1846–1858.

[125] Shailesh Kumar and Risto Miikkulainen. 1997. Dual reinforcement Q-routing: An on-line adaptive routing algorithm. In *Proceedings of the artificial neural networks in engineering Conference*. 231–238.

[126] Yongin Kwon, Sangmin Lee, Hayoon Yi, Donghyun Kwon, Seungjun Yang, Byung-Gon Chun, Ling Huang, Petros Maniatis, Mayur Naik, and Yunheung Paek. 2013. Mantis: Automatic performance prediction for smartphone applications. In *Presented as part of the 2013 {USENIX} Annual Technical Conference ({USENIX}{ATC} 13)*. 297–308.

[127] Marie-Anne Lachaux, Baptiste Roziere, Lowik Chanussot, and Guillaume Lample. 2020. Unsupervised Translation of Programming Languages. *arXiv preprint arXiv:2006.03511* (2020).

[128] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. 2015. Deep learning. *nature* 521, 7553 (2015), 436.

[129] Benjamin C Lee and David Brooks. 2010. Applied inference: Case studies in microarchitectural design. *ACM Transactions on Architecture and Code Optimization (TACO)* 7, 2 (2010), 8.

[130] Benjamin C Lee and David M Brooks. 2006. Accurate and efficient regression modeling for microarchitectural performance and power prediction. In *ACM SIGOPS Operating Systems Review*, Vol. 40. ACM, 185–194.

[131] Benjamin C Lee and David M Brooks. 2007. Illustrative design space studies with microarchitectural regression models. In *2007 IEEE 13th International Symposium on High Performance Computer Architecture*. IEEE, 340–351.

[132] Benjamin C Lee and David M Brooks. 2007. Spatial sampling and regression strategies. *IEEE Micro* 27, 3 (2007), 74–93.

[133] Benjamin C Lee, David M Brooks, Bronis R de Supinski, Martin Schulz, Karan Singh, and Sally A McKee. 2007. Methods of inference and learning for performance modeling of parallel applications. In *Proceedings of the 12th ACM SIGPLAN symposium on Principles and practice of parallel programming*. ACM, 249–258.

[134] Benjamin C Lee, Jamison Collins, Hong Wang, and David Brooks. 2008. CPR: Composable performance regression for scalable multiprocessor models. In *Proceedings of the 41st annual IEEE/ACM International Symposium on Microarchitecture*. IEEE Computer Society, 270–281.

[135] Jing Li, Xinpu Ji, Yuhan Jia, Bingpeng Zhu, Gang Wang, Zhongwei Li, and Xiaoguang Liu. 2014. Hard drive failure prediction using classification and regression trees. In *2014 44th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*. IEEE, 383–394.

[136] Jing Li, Rebecca J Stones, Gang Wang, Xiaoguang Liu, Zhongwei Li, and Ming Xu. 2017. Hard drive failure prediction using Decision Trees. *Reliability Engineering & System Safety* 164 (2017), 55–65.

[137] Yaguang Li, Yishuang Lin, Meghna Madhusudan, Arvind Sharma, Wenbin Xu, Sachin S Sapatnekar, Ramesh Harjani, and Jiang Hu. 2020. A customized graph neural network model for guiding analog IC placement. In *2020 IEEE/ACM International Conference On Computer Aided Design (ICCAD)*. IEEE, 1–9.

[138] Yunfan Li, D Penney, Abhishek Ramamurthy, and Lizhong Chen. 2019. Characterizing On-Chip Traffic Patterns in General-Purpose GPUs: A Deep Learning Approach. In *International Conference on Computer Design (ICCD)*.

[139] Shih-wei Liao, Tzu-Han Hung, Donald Nguyen, Chinyen Chou, Chiaheng Tu, and Hucheng Zhou. 2009. Machine learning-based prefetch optimization for data center applications. In *Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis*. ACM, 56.

[140] Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. 2015. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971* (2015).

[141] Ting-Ru Lin, Yunfan Li, Massoud Pedram, and Lizhong Chen. 2019. Design Space Exploration of Memory Controller Placement in Throughput Processors with Deep Learning. *IEEE Computer Architecture Letters* 18, 1 (2019), 51–54.

[142] Ting-Ru Lin, Drew Penney, Massoud Pedram, and Lizhong Chen. 2019. Optimizing Routerless Network-on-Chip Designs: An Innovative Learning-Based Framework. *arXiv preprint arXiv:1905.04423* (2019).

[143] Yibo Lin, Shounak Dhar, Wuxi Li, Haoxing Ren, Brucek Khailany, and David Z Pan. 2019. DREAMPIace: Deep Learning Toolkit-Enabled GPU Acceleration for Modern VLSI Placement. In *2019 56th ACM/IEEE Design Automation Conference (DAC)*. IEEE, 1–6.

[144] Nick Littlestone and Manfred K Warmuth. 1994. The weighted majority algorithm. *Information and computation* 108, 2 (1994), 212–261.

[145] Daniel Lo, Taejoon Song, and G Edward Suh. 2015. Prediction-guided performance-energy trade-off for interactive applications. In *Proceedings of the 48th International Symposium on Microarchitecture*. ACM, 508–520.

[146] Jonathan Long, Evan Shelhamer, and Trevor Darrell. 2015. Fully convolutional networks for semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 3431–3440.

[147] Shiting Justin Lu, Russell Tessier, and Wayne Burleson. 2015. Reinforcement learning for thermal-aware many-core task allocation. In *Proceedings of the 25th edition on Great Lakes Symposium on VLSI*. ACM, 379–384.

[148] Junshui Ma, Robert P Sheridan, Andy Liaw, George E Dahl, and Vladimir Svetnik. 2015. Deep neural nets as a method for quantitative structure–activity relationships. *Journal of chemical information and modeling* 55, 2 (2015), 263–274.

[149] Kai Ma, Xue Li, Wei Chen, Chi Zhang, and Xiaorui Wang. 2012. Greengpu: A holistic approach to energy efficiency in gpu-cpu heterogeneous architectures. In *2012 41st International Conference on Parallel Processing*. IEEE, 48–57.

[150] Peter S Magnusson, Magnus Christensson, Jesper Eskilson, Daniel Forsgren, Gustav Hallberg, Johan Hogberg, Fredrik Larsson, Andreas Moestedt, and Bengt Werner. 2002. Simics: A full system simulation platform. *Computer* 35, 2 (2002), 50–58.

[151] Farzaneh Mahdisoltani, Ioan Stefanovici, and Bianca Schroeder. 2017. Proactive error prediction to improve storage system reliability. In *2017 {USENIX} Annual Technical Conference ({USENIX}{ATC} 17)*. 391–402.

[152] Mateusz Majer, Christophe Bobda, Ali Ahmadinia, and Jürgen Teich. 2005. Packet routing in dynamically changing networks on chip. In *19th IEEE International Parallel and Distributed Processing Symposium*. IEEE, 8–pp.

[153] Hongzi Mao, Ravi Netravali, and Mohammad Alizadeh. 2017. Neural adaptive video streaming with pensieve. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*. 197–210.

[154] Artemiy Margaritov, Dmitrii Ustiugov, Edouard Bugnion, and Boris Grot. 2018. Virtual Address Translation via Learned Page Table Indexes. In *32nd Conference on Neural Information Processing Systems (NeurIPS)*.

[155] Jose F Martinez and Engin Ipek. 2009. Dynamic multicore resource management: A machine learning approach. *IEEE micro* 29, 5 (2009), 8–17.

[156] Amy McGovern, Eliot Moss, and Andrew G Barto. 2002. Building a basic block instruction scheduler with reinforcement learning and rollouts. *Machine learning* 49, 2-3 (2002), 141–160.

[157] Amy McGovern and J Eliot B Moss. 1999. Scheduling straight-line code using reinforcement learning and rollouts. In *Advances in Neural Information Processing Systems*. 903–909.

[158] Charith Mendis, Alex Renda, Saman Amarasinghe, and Michael Carbin. 2019. Ithemal: Accurate, Portable and Fast Basic Block Throughput Estimation using Deep Neural Networks. In *International Conference on Machine Learning*. 4505–4515.

[159] Charith Mendis, Cambridge Yang, Yewen Pu, Saman Amarasinghe, and Michael Carbin. 2019. Compiler Auto-Vectorization with Imitation Learning. In *Advances in Neural Information Processing Systems*. 14598–14609.

[160] Azalia Mirhoseini, Anna Goldie, Hieu Pham, Benoit Steiner, Quoc V Le, and Jeff Dean. 2018. A hierarchical model for device placement. In *Proceedings of the 35th International Conference on Machine Learning*. JMLR. org.

[161] Azalia Mirhoseini, Anna Goldie, Mustafa Yazgan, Joe Jiang, Ebrahim Songhori, Shen Wang, Young-Joon Lee, Eric Johnson, Omkar Pathak, Sungmin Bae, et al. 2020. Chip placement with deep reinforcement learning. *arXiv preprint arXiv:2004.10746* (2020).

[162] Azalia Mirhoseini, Hieu Pham, Quoc V Le, Benoit Steiner, Rasmus Larsen, Yuefeng Zhou, Naveen Kumar, Mohammad Norouzi, Samy Bengio, and Jeff Dean. 2017. Device placement optimization with reinforcement learning. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*. JMLR. org, 2430–2439.

[163] Nikita Mishra, Connor Imes, John D Lafferty, and Henry Hoffmann. 2018. CALOREE: Learning Control for Predictable Latency and Low Energy. In *Proceedings of the Twenty-Third International Conference on Architectural Support for Programming Languages and Operating Systems*. 184–198.

[164] Nikita Mishra, John D Lafferty, and Henry Hoffmann. 2017. Esp: A machine learning approach to predicting application interference. In *2017 IEEE International Conference on Autonomic Computing (ICAC)*. IEEE, 125–134.

[165] Nikita Mishra, Huazhe Zhang, John D Lafferty, and Henry Hoffmann. 2015. A probabilistic graphical model-based approach for minimizing energy under performance constraints. In *ACM SIGPLAN Notices*, Vol. 50. ACM, 267–281.

[166] Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. 2016. Asynchronous methods for deep reinforcement learning. In *International conference on machine learning*. 1928–1937.

[167] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. 2013. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602* (2013).

[168] Janani Mukundan and Jose F Martinez. 2012. MORSE: Multi-objective reconfigurable self-optimizing memory scheduler. In *IEEE International Symposium on High-Performance Comp Architecture*. IEEE, 1–12.

[169] Joseph F Murray, Gordon F Hughes, and Kenneth Kreutz-Delgado. 2005. Machine learning methods for predicting failures in hard drives: A multiple-instance application. *Journal of Machine Learning Research* 6, May (2005), 783–816.

[170] Shi Na, Liu Xumin, and Guan Yong. 2010. Research on k-means clustering algorithm: An improved k-means clustering algorithm. In *2010 Third International Symposium on intelligent information technology and security informatics*. IEEE, 63–67.

[171] Arvind Narayanan, Saurabh Verma, Eman Ramadan, Pariya Babaie, and Zhi-Li Zhang. 2018. Deepcache: A deep learning based framework for content caching. In *Proceedings of the 2018 Workshop on Network Meets AI & ML*. 48–53.

[172] Daniel Nemirovsky, Tugberk Arkose, Nikola Markovic, Mario Nemirovsky, Osman Unsal, and Adrian Cristal. 2017. A machine learning approach for performance prediction and scheduling on heterogeneous CPUs. In *2017 29th International Symposium on Computer Architecture and High Performance Computing (SBAC-PAD)*. IEEE, 121–128.

[173] Alex Nichol, Joshua Achiam, and John Schulman. 2018. On first-order meta-learning algorithms. *arXiv preprint arXiv:1803.02999* (2018).

[174] Kenneth O'Neal, Philip Brisk, Emily Shriver, and Michael Kishinevsky. 2017. HALWPE: Hardware-assisted light weight performance estimation for GPUs. In *2017 54th ACM/EDAC/IEEE Design Automation Conference (DAC)*. IEEE, 1–6.

[175] Berkin Ozisikyilmaz, Gokhan Memik, and Alok Choudhary. 2008. Machine learning models to predict performance of computer system design alternatives. In *2008 37th International Conference on Parallel Processing*. IEEE, 495–502.

[176] Meltem Ozsoy, Khaled N Khasawneh, Caleb Donovick, Iakov Gorelik, Nael Abu-Ghazaleh, and Dmitry Ponomarev. 2016. Hardware-based malware detection using low-level architectural features. *IEEE Trans. Comput.* 65, 11 (2016), 3332–3344.

[177] Gung-Yu Pan, Jing-Yang Jou, and Bo-Cheng Lai. 2014. Scalable power management using multilevel reinforcement learning for multiprocessors. *ACM Transactions on Design Automation of Electronic Systems (TODAES)* 19, 4 (2014), 33.

[178] Angshuman Parashar, Minsoo Rhu, Anurag Mukkara, Antonio Puglielli, Rangharajan Venkatesan, Brucek Khailany, Joel Emer, Stephen W Keckler, and William J Dally. 2017. Scnn: An accelerator for compressed-sparse convolutional neural networks. In *2017 ACM/IEEE 44th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 27–40.

[179] David Patterson, Thomas Anderson, Neal Cardwell, Richard Fromm, Kimberly Keeton, Christoforos Kozyrakis, Randi Thomas, and Katherine Yelick. 1997. A case for intelligent RAM. *IEEE micro* 17, 2 (1997), 34–44.

[180] Ashutosh Pattnaik, Xulong Tang, Adwait Jog, Onur Kayiran, Asit K Mishra, Mahmut T Kandemir, Onur Mutlu, and Chita R Das. 2016. Scheduling techniques for GPU architectures with processing-in-memory capabilities. In *Proceedings of the 2016 International Conference on Parallel Architectures and Compilation*. ACM, 31–44.

[181] Sai Manoj PD, Hao Yu, Hantao Huang, and Dongjun Xu. 2015. A Q-learning based self-adaptive I/O communication for 2.5 D integrated many-core microprocessor and memory. *IEEE Trans. Comput.* 65, 4 (2015), 1185–1196.

[182] Leeor Peled, Shie Mannor, Uri Weiser, and Yoav Etsion. 2015. Semantic locality and context-based prefetching using reinforcement learning. In *2015 ACM/IEEE 42nd Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 285–297.

[183] John C Platt, Emre Kıcıman, and David A Maltz. 2007. Fast variational inference for Large-scale Internet diagnosis. In *Proceedings of the 20th International Conference on Neural Information Processing Systems*. 1169–1176.

[184] Zhiliang Qian, Da-Cheng Juan, Paul Bogdan, Chi-Ying Tsui, Diana Marculescu, and Radu Marculescu. 2013. Svr-noc: A performance analysis tool for network-on-chips using learning-based support vector regression model. In *2013 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 354–357.

[185] Saami Rahman, Martin Burtscher, Ziliang Zong, and Apan Qasem. 2015. Maximizing hardware prefetch effectiveness with machine learning. In *2015 IEEE 17th International Conference on High Performance Computing and Communications,*

2015 IEEE 7th International Symposium on Cyberspace Safety and Security, and 2015 IEEE 12th International Conference on Embedded Software and Systems. IEEE, 383–389.

[186] Nishant Rao, Akshay Ramachandran, and Amish Shah. 2018. MLNoC: A Machine Learning based approach to NoC design. In *2018 30th International Symposium on Computer Architecture and High Performance Computing (SBAC-PAD)*. IEEE, 1–8.

[187] Gokul Subramanian Ravi and Mikko H Lipasti. 2017. CHARSTAR: Clock hierarchy aware resource scaling in tiled architectures. In *2017 ACM/IEEE 44th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 147–160.

[188] Veselin Raychev, Martin Vechev, and Eran Yahav. 2014. Code completion with statistical language models. In *Acm Sigplan Notices*, Vol. 49. ACM, 419–428.

[189] Sherief Reda, Ryan Cochran, and Ayse K Coskun. 2012. Adaptive power capping for servers with multithreaded workloads. *IEEE Micro* 32, 5 (2012), 64–75.

[190] Haoxing Ren, George F Kokai, Walker J Turner, and Ting-Sheng Ku. 2020. ParaGraph: Layout parasitics and device parameter prediction using graph neural networks. In *2020 57th ACM/IEEE Design Automation Conference (DAC)*. IEEE, 1–6.

[191] Alex Renda, Yishen Chen, Charith Mendis, and Michael Carbin. 2020. DiffTune: Optimizing CPU Simulator Parameters with Learned Differentiable Surrogates. *arXiv preprint arXiv:2010.04017* (2020).

[192] Md Farhadur Reza, Tung Thanh Le, Bappaditya De, Magdy Bayoumi, and Dan Zhao. 2018. Neuro-NoC: Energy optimization in heterogeneous many-core NoC using neural networks in dark silicon era. In *2018 IEEE International Symposium on Circuits and Systems (ISCAS)*. IEEE, 1–5.

[193] Nilabja Roy, Abhishek Dubey, and Aniruddha Gokhale. 2011. Efficient autoscaling in the cloud using predictive models for workload forecasting. In *2011 IEEE 4th International Conference on Cloud Computing*. IEEE, 500–507.

[194] MF Sakr, Steven P Levitan, Donald M Chiarulli, Bill G Horne, and C Lee Giles. 1997. Predicting multiprocessor memory access patterns with learning models. In *ICML*. 305–312.

[195] Ahmad EL Sallab, Mohammed Abdou, Etienne Perot, and Senthil Yogamani. 2017. Deep reinforcement learning framework for autonomous driving. *Electronic Imaging* 2017, 19 (2017), 70–76.

[196] Karthik Sangaiah, Mark Hempstead, and Baris Taskin. 2015. Uncore rpd: Rapid design space exploration of the uncore via regression modeling. In *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design*. IEEE Press, 365–372.

[197] Andreas G Savva, Theocharis Theocharides, and Vassos Soteriou. 2012. Intelligent on/off dynamic link management for on-chip networks. *Journal of Electrical and Computer Engineering* 2012 (2012), 6.

[198] Marwin HS Segler, Thierry Kogej, Christian Tyrchan, and Mark P Waller. 2017. Generating focused molecule libraries for drug discovery with recurrent neural networks. *ACS central science* 4, 1 (2017), 120–131.

[199] Keertana Settaluri, Ameer Haj-Ali, Qijing Huang, Kourosh Hakhamaneshi, and Borivoje Nikolic. 2020. Autockt: Deep reinforcement learning of analog circuit designs. In *2020 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 490–495.

[200] Ali Shafiee, Anirban Nag, Naveen Muralimanohar, Rajeev Balasubramonian, John Paul Strachan, Miao Hu, R Stanley Williams, and Vivek Srikumar. 2016. ISAAC: A convolutional neural network accelerator with in-situ analog arithmetic in crossbars. *ACM SIGARCH Computer Architecture News* 44, 3 (2016), 14–26.

[201] Zhan Shi, Xiangru Huang, Akanksha Jain, and Calvin Lin. 2019. Applying Deep Learning to the Cache Replacement Problem. In *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture*. ACM, 413–425.

[202] Zhan Shi, Akanksha Jain, Kevin Swersky, Milad Hashemi, Parthasarathy Ranganathan, and Calvin Lin. [n.d.]. A Neural Hierarchical Sequence Model for Irregular Data Prefetching. ([n. d.]).

[203] Zhan Shi, Kevin Swersky, Daniel Tarlow, Parthasarathy Ranganathan, and Milad Hashemi. 2019. Learning Execution through Neural Code Fusion. *arXiv preprint arXiv:1906.07181* (2019).

[204] Dongjoo Shin, Jinmook Lee, Jinsu Lee, and Hoi-Jun Yoo. 2017. 14.2 DNPU: An 8.1 TOPS/W reconfigurable CNN-RNN processor for general-purpose deep neural networks. In *2017 IEEE International Solid-State Circuits Conference (ISSCC)*. IEEE, 240–241.

[205] Brett Shook, Prateek Bhansali, Chandramouli Kashyap, Chirayu Amin, and Siddhartha Joshi. 2020. MLParest: machine learning based parasitic estimation for custom circuit design. In *2020 57th ACM/IEEE Design Automation Conference (DAC)*. IEEE, 1–6.

[206] David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, et al. 2017. Mastering the game of go without human knowledge. *nature* 550, 7676 (2017), 354–359.

[207] Karen Simonyan and Andrew Zisserman. 2014. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556* (2014).

[208] Vassos Soteriou, Theocharis Theocharides, and Elena Kakoulli. 2015. A holistic approach towards intelligent hotspot prevention in network-on-chip-based multicores. *IEEE Trans. Comput.* 65, 3 (2015), 819–833.

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

[210] Ilya Sutskever, Oriol Vinyals, and Quoc V Le. 2014. Sequence to sequence learning with neural networks. In *Advances in neural information processing systems*. 3104–3112.

[211] Richard S Sutton and Andrew G Barto. 2018. *Reinforcement learning: An introduction.* MIT press.

[212] Vivienne Sze, Yu-Hsin Chen, Tien-Ju Yang, and Joel S Emer. 2017. Efficient processing of deep neural networks: A tutorial and survey. *Proc. IEEE* 105, 12 (2017), 2295–2329.

[213] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. 2015. Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 1–9.

[214] Adrian Tang, Simha Sethumadhavan, and Salvatore J Stolfo. 2014. Unsupervised anomaly-based malware detection using hardware features. In *International Workshop on Recent Advances in Intrusion Detection*. Springer, 109–129.

[215] Stephen J Tarsa, Rangeen Basu Roy Chowdhury, Julien Sebot, Gautham Chinya, Jayesh Gaur, Karthik Sankaranarayanan, Chit-Kwan Lin, Robert Chappell, Ronak Singhal, and Hong Wang. 2019. Post-silicon cpu adaptation made practical using machine learning. In *Proceedings of the 46th International Symposium on Computer Architecture*. ACM, 14–26.

[216] Stephen J Tarsa, Chit-Kwan Lin, Gokce Keskin, Gautham Chinya, and Hong Wang. 2019. Improving Branch Prediction By Modeling Global History with Convolutional Neural Networks. *arXiv preprint arXiv:1906.09889* (2019).

[217] Elvira Teran, Zhe Wang, and Daniel A Jiménez. 2016. Perceptron learning for reuse prediction. In *2016 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 1–12.

[218] Gerald Tesauro. 2007. Reinforcement learning in autonomic computing: A manifesto and case studies. *IEEE Internet Computing* 11, 1 (2007), 22–30.

[219] Gerald Tesauro et al. 2005. Online resource allocation using decompositional reinforcement learning. In *AAAI*, Vol. 5. 886–891.

[220] Dana Van Aken, Andrew Pavlo, Geoffrey J Gordon, and Bohan Zhang. 2017. Automatic database management system tuning through large-scale machine learning. In *Proceedings of the 2017 ACM International Conference on Management of Data*. ACM, 1009–1024.

[221] Scott Van Winkle, Avinash Karanth Kodi, Razvan Bunescu, and Ahmed Louri. 2018. Extending the power-efficiency and performance of photonic interconnects for heterogeneous multicores with machine learning. In *2018 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 480–491.

[222] Joaquin Vanschoren. 2018. Meta-learning: A survey. *arXiv preprint arXiv:1810.03548* (2018).

[223] David Vengerov. 2009. A reinforcement learning framework for utility-based scheduling in resource-constrained systems. *Future Generation Computer Systems* 25, 7 (2009), 728–736.

[224] David Vengerov, Hamid R Berenji, and Alex Vengerov. 2002. Adaptive coordination among fuzzy reinforcement learning agents performing distributed dynamic load balancing. In *2002 IEEE World Congress on Computational Intelligence. 2002 IEEE International Conference on Fuzzy Systems. FUZZ-IEEE'02. Proceedings (Cat. No. 02CH37291)*, Vol. 1. IEEE, 179–184.

[225] David Vengerov and Nikolai Iakovlev. 2005. A Reinforcement Learning Framework for Dynamic Resource Allocation: First Results.. In *Second International Conference on Autonomic Computing (ICAC'05)*. IEEE, 339–340.

[226] Oriol Vinyals, Igor Babuschkin, Wojciech M Czarnecki, Michaël Mathieu, Andrew Dudzik, Junyoung Chung, David H Choi, Richard Powell, Timo Ewalds, Petko Georgiev, et al. 2019. Grandmaster level in StarCraft II using multi-agent reinforcement learning. *Nature* 575, 7782 (2019), 350–354.

[227] Boqian Wang, Zhonghai Lu, and Shenggang Chen. 2019. ANN Based Admission Control for On-Chip Networks. In *Proceedings of the 56th Annual Design Automation Conference 2019*. ACM, 46.

[228] Haoyuan Wang and Zhiwei Luo. 2017. Data Cache Prefetching with Perceptron Learning. *arXiv preprint arXiv:1712.00905* (2017).

[229] Hanrui Wang, Kuan Wang, Jiacheng Yang, Linxiao Shen, Nan Sun, Hae-Seung Lee, and Song Han. 2020. GCN-RL circuit designer: Transferable transistor sizing with graph neural networks and reinforcement learning. In *2020 57th ACM/IEEE Design Automation Conference (DAC)*. IEEE, 1–6.

[230] Hua Wang, Xinbo Yi, Ping Huang, Bin Cheng, and Ke Zhou. 2018. Efficient SSD Caching by Avoiding Unnecessary Writes using Machine Learning. In *Proceedings of the 47th International Conference on Parallel Processing*. ACM, 82.

[231] Ke Wang, Ahmed Louri, Avinash Karanth, and Razvan Bunescu. 2019. High-performance, Energy-efficient, Fault-tolerant Network-on-Chip Design Using Reinforcement Learnin. In *2019 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 1166–1171.

[232] Ke Wang, Ahmed Louri, Avinash Karanth, and Razvan Bunescu. 2019. IntelliNoC: a holistic design framework for energy-efficient and reliable on-chip communication for manycores. In *Proceedings of the 46th International Symposium*

on Computer Architecture. ACM, 589–600.

[233] Shibo Wang and Engin Ipek. 2016. Reducing data movement energy via online data clustering and encoding. In *The 49th Annual IEEE/ACM International Symposium on Microarchitecture*. IEEE Press, 32.

[234] Rainer Waser, Regina Dittmann, Georgi Staikov, and Kristof Szot. 2009. Redox-based resistive switching memories–nanoionic mechanisms, prospects, and challenges. *Advanced materials* 21, 25-26 (2009), 2632–2663.

[235] Christopher JCH Watkins and Peter Dayan. 1992. Q-learning. *Machine learning* 8, 3-4 (1992), 279–292.

[236] Shimon Whiteson and Peter Stone. 2004. Adaptive job routing and scheduling. *Engineering Applications of Artificial Intelligence* 17, 7 (2004), 855–869.

[237] Ronald J Williams. 1992. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning* 8, 3-4 (1992), 229–256.

[238] Svante Wold, Kim Esbensen, and Paul Geladi. 1987. Principal component analysis. *Chemometrics and intelligent laboratory systems* 2, 1-3 (1987), 37–52.

[239] Jae-Yeon Won, Xi Chen, Paul Gratz, Jiang Hu, and Vassos Soteriou. 2014. Up by their bootstraps: Online learning in artificial neural networks for CMP uncore power management. In *2014 IEEE 20th International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 308–319.

[240] Gene Wu, Joseph L Greathouse, Alexander Lyashevsky, Nuwan Jayasena, and Derek Chiou. 2015. GPGPU performance and power estimation using machine learning. In *2015 IEEE 21st International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 564–576.

[241] Gang Wu, Yue Xu, Dean Wu, Manoj Ragupathy, Yu-yen Mo, and Chris Chu. 2016. Flip-flop clustering by weighted K-means algorithm. In *Proceedings of the 53rd Annual Design Automation Conference*. ACM, 82.

[242] Nan Wu, Lei Deng, Guoqi Li, and Yuan Xie. 2020. Core Placement Optimization for Multi-chip Many-core Neural Network Systems with Reinforcement Learning. *ACM Transactions on Design Automation of Electronic Systems (TODAES)* 26, 2 (2020), 1–27.

[243] Nan Wu and Pengcheng Li. 2020. Phoebe: Reuse-Aware Online Caching with Reinforcement Learning for Emerging Storage Models. *arXiv preprint arXiv:2011.07160* (2020).

[244] Weidan Wu and Benjamin C Lee. 2012. Inferred models for dynamic and sparse hardware-software spaces. In *Proceedings of the 2012 45th Annual IEEE/ACM International Symposium on Microarchitecture*. IEEE Computer Society, 413–424.

[245] Jiang Xiao, Zhuang Xiong, Song Wu, Yusheng Yi, Hai Jin, and Kan Hu. 2018. Disk failure prediction in data centers via online learning. In *Proceedings of the 47th International Conference on Parallel Processing*. ACM, 35.

[246] Chang Xu, Gang Wang, Xiaoguang Liu, Dongdong Guo, and Tie-Yan Liu. 2016. Health status assessment and failure prediction for hard drives with recurrent neural networks. *IEEE Trans. Comput.* 65, 11 (2016), 3502–3508.

[247] Yong Xu, Kaixin Sui, Randolph Yao, Hongyu Zhang, Qingwei Lin, Yingnong Dang, Peng Li, Keceng Jiang, Wenchi Zhang, Jian-Guang Lou, et al. 2018. Improving service availability of cloud systems by predicting disk error. In *2018 {USENIX} Annual Technical Conference ({USENIX}{ATC} 18)*. 481–494.

[248] Amir Yazdanbakhsh, Jongse Park, Hardik Sharma, Pejman Lotfi-Kamran, and Hadi Esmaeilzadeh. 2015. Neural acceleration for GPU throughput processors. In *Proceedings of the 48th International Symposium on Microarchitecture*. ACM, 482–493.

[249] Hyunho Yeo, Youngmok Jung, Jaehong Kim, Jinwoo Shin, and Dongsu Han. 2018. Neural adaptive content-aware internet video delivery. In *13th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 18)*. 645–661.

[250] Nezih Yigitbasi, Theodore L Willke, Guangdeng Liao, and Dick Epema. 2013. Towards machine learning-based auto-tuning of mapreduce. In *2013 IEEE 21st International Symposium on Modelling, Analysis and Simulation of Computer and Telecommunication Systems*. IEEE, 11–20.

[251] Jieming Yin, Yasuko Eckert, Shuai Che, Mark Oskin, and Gabriel H Loh. 2018. Toward More Efficient NoC Arbitration: A Deep Reinforcement Learning Approach. (2018).

[252] Jieming Yin, Subhash Sethumurugan, Yasuko Eckert, Chintan Patel, Alan Smith, Eric Morton, Mark Oskin, Natalie Enright Jerger, and Gabriel H Loh. 2020. Experiences with ML-Driven Design: A NoC Case Study. In *2020 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 637–648.

[253] Zhenlong Yuan, Yongqiang Lu, Zhaoguo Wang, and Yibo Xue. 2014. Droid-sec: deep learning in android malware detection. In *ACM SIGCOMM Computer Communication Review*, Vol. 44. ACM, 371–372.

[254] Stephen Zekany, Daniel Rings, Nathan Harada, Michael A Laurenzano, Lingjia Tang, and Jason Mars. 2016. CrystalBall: Statically analyzing runtime behavior via deep sequence learning. In *2016 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 1–12.

[255] Yuan Zeng and Xiaochen Guo. 2017. Long short term memory based hardware prefetcher: A case study. In *Proceedings of the International Symposium on Memory Systems*. ACM, 305–311.

[256] Haitao Zhang, Bingchang Tang, Xin Geng, and Huadong Ma. 2018. Learning Driven Parallelization for Large-Scale Video Workload in Hybrid CPU-GPU Cluster. In *Proceedings of the 47th International Conference on Parallel Processing*. ACM, 32.

[257] Kaiqing Zhang, Zhuoran Yang, and Tamer Başar. 2019. Multi-agent reinforcement learning: A selective overview of theories and algorithms. *arXiv preprint arXiv:1911.10635* (2019).

[258] Hao Zheng and Ahmed Louri. 2019. An energy-efficient network-on-chip design using reinforcement learning. In *Proceedings of the 56th Annual Design Automation Conference 2019*. ACM, 47.

[259] Xinnian Zheng, Lizy K John, and Andreas Gerstlauer. 2016. Accurate phase-level cross-platform power and performance estimation. In *2016 53nd ACM/EDAC/IEEE Design Automation Conference (DAC)*. IEEE, 1–6.

[260] Xinnian Zheng, Pradeep Ravikumar, Lizy K John, and Andreas Gerstlauer. 2015. Learning-based analytical cross-platform performance prediction. In *2015 International Conference on Embedded Computer Systems: Architectures, Modeling, and Simulation (SAMOS)*. IEEE, 52–59.

[261] Yanqi Zhou, Sudip Roy, Amirali Abdolrashidi, Daniel Wong, Peter C Ma, Qiumin Xu, Ming Zhong, Hanxiao Liu, Anna Goldie, Azalia Mirhoseini, et al. 2019. GDP: Generalized Device Placement for Dataflow Graphs. *arXiv preprint arXiv:1910.01578* (2019).

[262] Bingpeng Zhu, Gang Wang, Xiaoguang Liu, Dianming Hu, Sheng Lin, and Jingwei Ma. 2013. Proactive drive failure prediction for large scale storage systems. In *2013 IEEE 29th symposium on mass storage systems and technologies (MSST)*. IEEE, 1–5.

[263] Xiaojin Jerry Zhu. 2005. *Semi-supervised learning literature survey.* Technical Report. University of Wisconsin-Madison Department of Computer Sciences.

[264] Cheng Zhuo, Bei Yu, and Di Gao. 2017. Accelerating chip design with machine learning: From pre-silicon to post-silicon. In *2017 30th IEEE International System-on-Chip Conference (SOCC)*. IEEE, 227–232.

[265] Fei Zuo, Xiaopeng Li, Patrick Young, Lannan Luo, Qiang Zeng, and Zhexin Zhang. 2018. Neural machine translation inspired binary code similarity comparison beyond function pairs. *arXiv preprint arXiv:1808.04706* (2018).