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 for efficient execution of machine learning (ML) 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: improvement of designers' productivity and completion of the virtuous cycle. In this article, we present a comprehensive review of the work that applies ML for computer architecture and system design. First, we perform a high-level taxonomy by considering the typical role that ML techniques take in architecture/system design, i.e., either for fast predictive modeling or as the design methodology. Then, we summarize the common problems in computer architecture/system design that can be solved by ML techniques and the typical ML techniques employed to resolve each of them. In addition to emphasis on computer architecture in a narrow sense, we adopt the concept that data centers can be recognized as warehouse-scale computers; sketchy discussions are provided in adjacent computer systems, such as code generation and compiler; we also give attention to how ML techniques can aid and transform 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, machine learning for systems

ACM Reference format:
Nan Wu and Yuan Xie. 2022. A Survey of Machine Learning for Computer Architecture and Systems. *ACM Comput. Surv.* 55, 3, Article 54 (February 2022), 39 pages.
https://doi.org/10.1145/3494523

1 INTRODUCTION

Machine learning (ML) has been doing wonders in many fields. As people are seeking better **artificial intelligence (AI)**, there is a trend towards larger, more expressive, and more complex models. According to the data reported by OpenAI [21], from 1959 to 2012, the amount of compute used in the largest AI training runs doubles every two years; since 2012, deep learning starts taking off, and the required amount of compute has been increasing exponentially with a 3.4-month doubling period. By comparison, Moore's law [168], the principle that has powered the integrated-circuit revolution since 1960s, doubles the transistor density every 18 months. While Moore's law

Authors' address: N. Wu and Y. Xie, Department of Electrical and Computer Engineering, University of California, Santa Barbara, CA, 93106, USA; emails: {nanwu, yuanxie}@ucsb.edu.

0360-0300/2022/02-ART54 $15.00
https://doi.org/10.1145/3494523

is approaching its end [228], more pressure is put on innovations of computer architecture and systems to keep up with the compute demand of AI applications.

Conventionally, computer architecture/system designs are made by human experts based on intuitions and heuristics, which requires expertise in both ML and architecture/system. Meanwhile, these heuristic-based designs can not guarantee scalability and optimality, especially in the case of increasingly complicated systems. As such, it seems natural to move towards more automated and powerful methodologies for computer architecture and system design, and the relationship between ML and system design is being reconsidered. Over the past decade, architecture and systems are optimized to accelerate the execution and improve the performance of ML models. Recently, there have been signs of emergence of applying ML for computer architecture and systems, which embraces a twofold meaning: ① the reduction of burdens on human experts designing systems manually to improve designers' productivity, and ② the close of the positive feedback loop, i.e., architecture/systems for ML and simultaneously ML for architecture/systems, formulating a virtuous cycle to encourage improvements on both sides.

Existing work related to applying ML for computer architecture and system design falls into two categories: ① ML techniques are employed for **fast and accurate system modeling**, which involves performance metrics or some criteria of interest (e.g., power consumption, latency, throughput). During the process of designing systems, it is necessary to make fast and accurate predictions of system behaviors. Traditionally, system modeling is achieved through the forms of cycle-accurate or functional virtual platforms and instruction set simulators (e.g., gem5 [18]). Even though these methods provide accurate estimations, they bring expensive computation costs associated with performance modeling, which limits the scalability to large-scale and complex systems; meanwhile, the long simulation time often dominates design iteration, making it impossible to fully explore the design space. By contrast, ML-based modeling and performance prediction are capable to balance simulation cost and prediction accuracy. ② ML techniques are employed as **a design methodology to directly enhance architecture/system design**. ML techniques are skilled at extracting features that might be implicit to human experts, making decisions without explicit programming, and improving themselves automatically with accumulated experience. Therefore, applying ML techniques as design tools can explore design space proactively and intelligently, and manage resource through better understanding of the complicated and non-linear interactions between workloads and systems, making it possible to deliver truly optimal solutions.

In this article, we present a comprehensive overview of applying ML for computer architecture and systems. As depicted in Figure 1, we first perform a high-level taxonomy by considering the typical role that ML techniques take in architecture/system design, i.e., either for fast predictive modeling or as the design methodology; then, we summarize the common problems in architecture/system design that can be solved by ML techniques and the typical ML techniques employed to resolve each of them. In addition to emphasis on computer architecture in a narrow sense, we adopt the concept that data centers can be recognized as warehouse-scale computers [15] and review studies associated with data center management; we provide sketchy discussions on adjacent computer systems, such as code generation and compiler; we also give attention to how ML techniques can aid and transform design automation that involves both analog and digital circuits. At the end of the article, we discuss challenges and future prospects of applying ML for architecture/system design, aiming to convey insights of design considerations.

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. Table 1 summarizes the commonly used ML

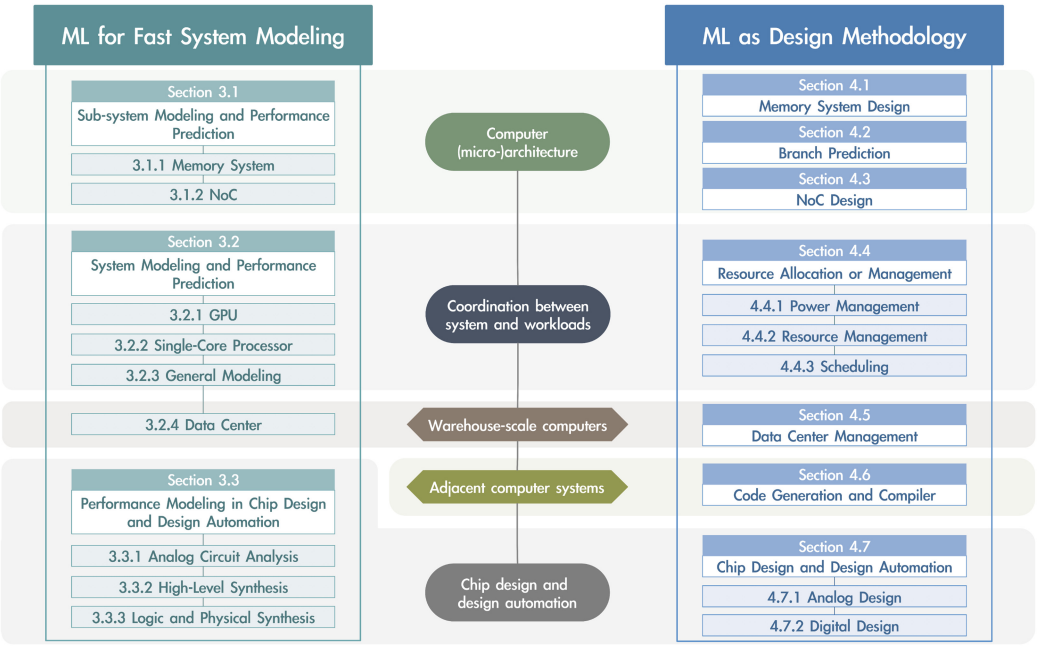

Fig. 1. A comprehensive overview of applying ML for computer architecture and systems. Existing work roughly falls into two categories: ML for fast system modeling and ML as design methodology.

Table 1. Machine Learning Techniques

Realm of ML	Category	Classical ML	Deep Learning Counterpart [71]
Supervised Learning	Classification	Logistic regression [80]	CNN, RNN, GNN [245], etc.
	Working for both	Support vector machines/regression [202]	
		K-nearest neighbors [8]	
		Decision tree, e.g., CART [141], MARS [62]	
		ANN [197]	
		Bayesian analysis [68]	
		Ensemble learning [199], e.g., gradient boosting, random forest	
	Regression	Linear regression with variants [204], e.g., lasso (L1 regularization), ridge (L2 regularization), elastic-net (hybrid L1/L2 regularization)	
		Non-linear regression [193]	
Unsupervised Learning	Clustering	K-means clustering [88]	Autoencoder, GAN, etc.
	Dimension reduction	Principal component analysis (PCA) [237]	
Reinforcement Learning	Value-based	Q-learning [214]	DQN [167]
	Policy-based	Actor-critic [214]	A3C [166], DDPG [132]
		Policy gradient, e.g., REINFORCE [214]	PPO [203]

techniques for computer architecture and system designs. Sometimes, multiple learning models may work well for one given problem, and the appropriate selection can be made based on available hardware resource and data, implementation overheads, performance targets, and so on.

2.1 Supervised Learning

Supervised learning is the process of learning a set of rules able to map an input to an output based on labeled datasets. These learned rules can be generalized to make predictions for unseen inputs. We briefly introduce several prevalent techniques in supervised learning, as shown in Figure 2.

- Regression is a process for estimating the relationships between a dependent variable and one or more independent variables. The most common form is linear regression [204], and some other forms include different types of non-linear regression [193]. Regression

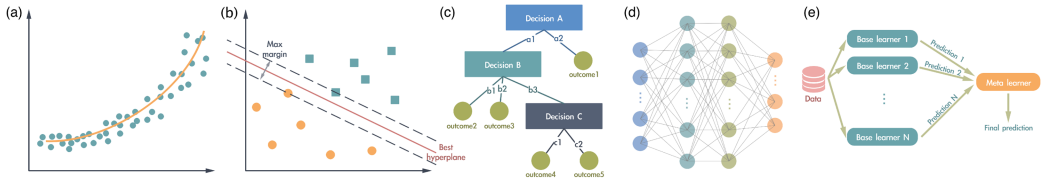

Fig. 2. Examples of supervised learning: (a) regression, (b) SVM, (c) decision tree, (d) MLP, and (e) ensemble learning.

techniques are primarily used for two purposes, prediction/forecasting, and inference of causal relationships.

- **Support vector machines (SVMs)** [202] try to find the best hyperplanes to separate data classes by maximizing margins. One variant is **support vector regression (SVR)**, which is able to conduct regression tasks. Predictions or classifications of new inputs can be decided by their relative positions to these hyperplanes.
- Decision tree is one representative of logical learning methods, which uses tree structures to build regression or classification models. The final result is a tree with decision nodes and leaf nodes. Each decision node represents a feature, and branches of this node represent possible values of the corresponding feature. Starting from the root node, input instances are classified by sequentially passing through nodes and branches, until they reach leaf nodes that represent either classification results or numerical values.
- **Artificial neural networks (ANNs)** [197] are capable to approximate a broad family of functions: a single-layer perceptron is usually used for linear regression; complex DNNs [71] 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 exploitation of certain computation operations, e.g., **convolutional neural networks (CNNs)** with convolution operations leveraging spatial features, and **recurrent neural networks (RNNs)** with recurrent connections enabling learning from sequences and histories.
- Ensemble learning [199] employs multiple models that are strategically designed to solve a particular problem, and the primary goal is to achieve better predictive performance than those could be obtained from any of the constituent models alone. Several common types of ensembles include random forest and gradient boosting.

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. One thing worth noting is that supervised learning techniques need well labeled training data prior to the training phase, which usually require tremendous human expertise and engineering.

2.2 Unsupervised Learning

Unsupervised learning is the process of finding previously unknown patterns based on unlabeled datasets. Two prevailing methods are clustering analysis [88] and **principal component analysis (PCA)** [237], as depicted in Figure 3.

- 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

Fig. 3. Examples of unsupervised learning: (a) clustering and (b) PCA.

Fig. 4. A typical framing of RL.

clusters share low similarities. The goal of clustering is to classify raw data reasonably and to find possibly existing hidden structures or patterns in datasets. One of the most popular and simple clustering algorithms is K-means clustering.

- 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.

Since there is no label in unsupervised learning, it is difficult to simultaneously measure the performance of learning models and decide when to stop the learning process. One potential workaround is *semi-supervised learning* [273], 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.

2.3 Reinforcement Learning

In standard **reinforcement learning (RL)** [214], an agent interacts with an environment $\mathcal{E}$ over a number of discrete time steps, as shown in Figure 4. 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 $r_t : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$. 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 totally accumulated rewards at the timestep 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.

- 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 always selects 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.
- 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.

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.

Table 2. Summary of Applying ML Techniques for Fast Modeling in Computer Architecture and Systems

Domain	Prediction of	Technique	Input
Memory System (Section 3.1.1)	Throughput/cache miss	ANN [52]	Cache configurations
	Throughput/lifetime/energy	Gradient boosting/quadratic regression with lasso [48]	NVM configurations
	Throughput	CNN [133]	Memory controller placements
	Disk block correlation	DNN [42]	Data blocks in the context window
NoC (Section 3.1.2)	Latency	SVR [185]	Queue information
	Hotspots	ANN [212]	Buffer utilization
	Buffer/link utilization	Regression with ridge regularization [35, 224], decision tree [51]	NoC configurations
	Probability of errors	Decision tree [50]	Link utilization and transistor wearing-out
GPU (Section 3.2.1)	Speedup or execution time by cross-platform inputs	Nearest neighbor/SVM [14], ensemble of regression-based learners [9], random forest [10]	Static analysis and/or dynamic profiling of source CPU code
	Execution time	Stepwise regression [90], ensemble of linear regression and random forest [175]	GPU configurations and performance counters
	Throughput/power	Ensemble of NNs [99]	
	Scaling behavior of GPGPU	ANN and K-means clustering [239]	
	Kernel affinity and execution time	Logistic regression and linear regression [181]	Kernel characteristics
	Traffic patterns in GPGPU	CNN [130]	Grayscale heat maps
Single-Core CPU (Section 3.2.2)	Throughput	Linear regression [100], non-linear regression [57, 101, 121]	Micro-architectural parameters and performance counters
	Program execution time	Linear regression [268, 269]	
General Modeling (Section 3.2.3)	Throughput/latency/power	ANN [19, 83, 110, 123, 172, 177]	Micro-architectural parameters and performance counters
		Non-linear regression [122, 124, 200, 244]	
		Linear regression [12, 41, 140]	
		Hierarchical Bayesian model [163, 165]	
		LSTM [157, 191]	
		Generative model [49]	
	Slowdown caused by application interference	Linear regression with elastic-net regularization [164]	
	Speedup of multi-thread applications	Gaussian process regression [3]	Profiling of single-thread execution
Data Center (Section 3.2.4)	Job completion time	SVR [253]	Application characteristics and cluster configurations
	Resource demand	Statistical learning [70], linear regression/MLP [85]	Workload characterization
	Incoming workload	ARMA [196], ARIMA [27]	
	Workload pattern	Hidden Markov model [109]	
	Power usage effectiveness	MLP [65]	Data center configurations
	Disk Failure	Bayesian methods [75], clustering [170], SVM/MLP [271], random forest [246]	SMART (Self-Monitoring, Analysis and Reporting Technology) attributes of data centers
	Health assessment of drives	CART [126], gradient boosted regressions tree [127], RNN [248]	
	Partial drive failure	CART/random forest/SVM/ANN/logistic regression [146]	
		Gradient boosted regression trees [249]	SMART attributes and system-level signals

3 ML FOR FAST SYSTEM MODELING

This section reviews studies that employ ML techniques for fast and accurate system modeling, which involves predictions of performance metrics or some other criteria of interest. Although cycle-accurate simulators, which are commonly used for system performance prediction, can provide accurate estimations, they usually run multiple orders of magnitude slower than native executions. By contrast, ML-based techniques can balance simulation costs and prediction accuracy, showing great potentials in exploring huge configuration spaces and learning non-linear impacts of configurations. Most of existing work applies supervised learning for either pure system modeling or efficient **design space exploration (DSE)** enabled by fast predictions. Tables 2 and 3 summarize the studies for predictive modeling in computer architecture/system and design automation, respectively, in terms of task domains, prediction targets, adopted ML techniques, and corresponding inputs.

3.1 Sub-system Modeling and Performance Prediction

3.1.1 Memory System. In memory systems, ML-based performance models are exploited to help explore tradeoffs among different objectives. To explore **non-volatile memory (NVM)**-based cache hierarchies, Dong et al. [52] develop an ANN model 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, Memory Cocktail Therapy [48] estimates lifetime, IPC, and energy

Table 3. Summary of Applying ML Techniques for Performance Modeling and Prediction in Design Automation

Domain	Prediction of	Technique	Input
Analog Circuit (Section 3.3.1)	Parasitics	GNN [190], random forest [210]	Circuit schematics
	Power/area/bandwidth	Bayesian co-learning and semi-supervised learning [5], Bayesian regression and SVM [179]	Circuit schematics and device information
	Probability of superiority between designs	Bayesian DNN [74]	
	Gain/unity gain frequency/bandwidth/phase margin	SVM/ANN/random forest [128], 3D CNN [138], GNN [129]	Circuit placement
	Electromagnetic properties	GNN [262]	
HLS (Section 3.3.2)	Area/latency/throughput/logic utilization	Random forest [137, 159], transfer learning [119]	Directives in HLS scripts
	Resource utilization	ANN [112]	IR graphs from HLS front-ends
	Resource mapping and clustering	GraphSAGE [223]	
	Routing congestion	Linear regression/ANN/gradient boosted regression tree [265]	
	Power	Linear regression/SVM/tree-based models/DNNs [136]	IR graphs and HLS reports
	Throughput and throughput-to-area ratio	Ensemble learning by stacked regression [148]	HLS reports
	Resource utilization and timing	Linear regression/ANN/gradient tree boosting [43]	
	Cross-platform latency and power	Random forest [176]	CPU program counters
	Speedup over an ARM processor	ANN [149]	Application characteristics, HLS reports, FPGA configurations
Logic and Physical Synthesis (Section 3.3.3)	Area/delay	CNN [256], LSTM [258]	Synthesis flows
	DRVs	Linear regression/ANN/decision tree [125], MARS [184]	Placement and GR information
		MARS/SVM [28], MLP [216]	Placement
	DRC hotspots	FCN [247]	Placement and GR information
		Variant of FCN [131]	Placement
	GR congestion map	FCN [29]	
	Routing congestion in FPGAs	Linear regression [145]	
		Conditional GAN [6, 257]	Post-placement images

consumption through lightweight online predictors by gradient boosting and quadratic regression with lasso. To optimize memory controller placements in throughput processors, Lin et al. [133] build a CNN model that takes memory controller placements as inputs to predict throughput, which accelerates the optimization process by two orders of magnitude.

Some studies concentrate on learning efficient representations of memory access patterns. Block2Vec [42] tries to mine data block correlations by training a DNN to learn the best vector representation of each block and capturing block similarities via vector distances, which enables further optimization for caching and prefetching. Shi et al. [209] use a **graph neural network (GNN)** to learn fused representations of static code and its dynamic execution. This unified representation is capable to model both data flows (e.g., prefetching) and control flows (e.g., branch prediction).

3.1.2 Network-on-chip (NoC). In NoCs, several performance metrics of interest are latency, energy consumption, and reliability. ① Regarding latency predictions, Qian et al. [185] use an SVR model to predict the traffic flow latency and the average channel waiting time in mesh-based NoCs, which relaxes some assumptions in the classical queuing theory. Rather than explicitly predicting latency, a lightweight hardware-based ANN [212] predicts existence of traffic hotspots, which are intensive network congestions significantly degrading the effective throughput and implicitly indicate the average communication latency in NoCs. The input features are buffer utilization rates from neighboring NoC routers, and the trained predictor is combined with a proactive hotspot-preventive routing algorithm to avert hotspot formation, attaining significant improvements for synthetic workloads while slight melioration for real-world benchmarks ② Regarding estimating energy consumption, the learned predictors are often leveraged for saving dynamic and/or static energy in NoCs. DiTomaso et al. [51] use per-router decision trees to predict link utilization and traffic direction, which are combined with sleepy link storage units to power-gate links/routers and to change link directions. Clark et al. [35] use ridge regression models to predict buffer utilization, changes in buffer utilization, or a combined metric of energy and throughput, based on which a router can select proper voltage/frequency. In photonic NoCs, the ridge regression model is also applicable to predict the number of packets to be injected into each router in the following time window [224], based on which the number of wavelengths are scaled properly to reduce

static energy consumed by photonic links. ③ Regarding the reliability of NoCs, a per-link decision tree trained offline can predict the probability of timing faults on links during runtime [50], based on which a proactive fault-tolerant technique is developed to mitigate errors by using the strengthened cyclic redundancy check with error-correction code and relaxed transmission.

3.2 System Modeling 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 can provide highly accurate performance estimations with reasonable simulation costs, surpassing the capability of commonly used cycle-accurate simulators that require highly computational costs and long simulation time.

3.2.1 Graphics Processing Unit (GPU). There are two types of predictions for GPU modeling: cross-platform predictions and GPU-specific predictions. Cross-platform predictions are used 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 considerably additional efforts; GPU-specific predictions are used to estimate metrics of interest and to assist GPU design space exploration, helpful to handle design space irregularities and complicated interactions among configurations.

Cross-platform predictions can be formulated as a binary classification problem that identifies whether the potential GPU speedup of an application would be greater than a given threshold. This task can be solved by the nearest neighbor and SVMs using dynamic instruction profiles [14] or a random forest composed of 1,000 decision trees using static analysis of source CPU code (i.e., memory coalescing, branch divergence, kernel size available parallelism, and instruction intensities) [10]. With both dynamic and static program properties from single-thread CPU code, an ensemble of 100 regression-based learners can predict the GPU execution time [9].

In terms of GPU-specific predictions that take GPU configurations and performance counters as input features, the execution time can be predicted by stepwise linear regression, which recognizes the most important input features among many GPU parameters and thus achieves high accuracy even with sparse samples [90]; the power/throughput can be modeled by an ensemble of NN predictors [99]. Provided with profiling results from earlier-generation GPUs, an ensemble of linear and non-linear regression models is capable to predict cross-generation GPU execution time for later/future-generation GPUs, which achieves more than 10,000 times speedup compared to cycle-accurate GPU simulators [175]. Focusing on **processing-in-memory (PIM)**-assisted GPU architectures, Pattnaik et al. [181] classify GPU cores into two types: powerful GPU cores but far away from memory, and auxiliary/simple GPU cores but close to memory. They develop a logistic regression model that takes kernel characteristics as input features to predict architecture affinity of kernels, aiming to accurately identify which kernels would benefit from PIM and offload them accordingly to auxiliary GPU cores. They also build a linear regression model to predict the execution time of each kernel, so a concurrent kernel management mechanism can be developed based on these two models and kernel dependency information. Focusing on **general-purpose GPUs (GPGPUs)**, Wu et al. [239] model kernel scaling behaviors with respect to the number of compute units, engine frequency, and memory frequency. During training, kernels with similar performance scaling behaviors are grouped by K-means clustering, and when encountering a new kernel, it is mapped to the cluster that best describes its scaling performance by an ANN-based classifier. Li et al. [130] reassess prevailing assumptions of GPGPU traffic patterns and combine a CNN with a t-distributed stochastic neighbor embedding to classify different traffic patterns.

3.2.2 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. [100] 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 [101]. In parallel with Joseph's work, Lee and Brooks [121] use regression modeling with cubic splines to predict application-specific performance (billions of instructions per second) and power.

Later work focuses on performance modeling for existing hardware (e.g., Intel, AMD, and ARM processors) by using micro-architectural parameters and performance counters. Eyerman et al. [57] construct a mechanistic-empirical model for CPI predictions of three Intel processors. The initially parameterized performance model is inspired by mechanistic modeling, where the unknown parameters inside the model are derived through regression, benefiting from both mechanistic modeling (i.e., interpretability) and empirical modeling (i.e., ease of implementation). Zheng et al. [268, 269] explore two approaches to cross-platform predictions of program execution time, where profiling results on Intel Core i7 and AMD Phenom processors are used to estimate the execution time on a target ARM processor. The first approach [269] relaxes the assumption of global linearity to local linearity in the feature space and applies constrained locally sparse linear regression; the other approach [268] applies lasso linear regression with phase-level performance features.

3.2.3 General Modeling and Performance Prediction. Regression techniques are the mainstream to predict performance metrics from micro-architectural parameters or other features, which attributes to their capability to make high-accuracy estimations with reasonable training costs.

For conventional regression-based models, ANNs and non-linear regression with different designs are the common practice to predict throughput/latency [83, 110, 122, 124, 244] and power/energy [110, 122]. Consequently, there are comparisons among different techniques. Lee et al. [123] compare piecewise polynomial regression with ANNs, with emphasis that piecewise polynomial regression offers better explainability, while ANNs show better generalization ability. Ozisikyilmaz et al. [177] contrast several linear regression models and different ANNs, indicating that the pruned ANNs achieve best accuracy while requiring longer training time. Agarwal et al. [3] estimate the parallel execution speedup of multi-threaded applications on a target hardware platform and mention that Gaussian process regression performs the best among several explored methods in this case.

More recent work tends to take advantage of data-driven approaches. Ithemal [157] leverages a hierarchical multi-scale RNN with **long short term memory (LSTM)** to predict throughput of basic blocks (i.e., sequences of instructions with no branches or jumps), and evaluations demonstrate that Ithemal is more accurate and as fast as analytical throughput estimators. By employing a variant of Ithemal as a differentiable surrogate to approximate CPU simulators, DiffTune [191] is able to apply gradient-based optimization techniques to learn the parameters of x86 basic block CPU simulators such that simulators' error is minimized. The learned parameters finally are plugged back into the original simulator. Ding et al. [49] provide some insights in learning-based modeling methods: the improvement of prediction accuracy may receive diminishing returns; the consideration of domain knowledge will be helpful for system optimizations, even if the overall accuracy may not be improved. Thus, they propose a generative model to handle data scarcity by generating more training data and apply a multi-phase sampling to improve prediction accuracy of optimal configuration points.

ML-based predictive performance modeling enables efficient resource management and rapid design space exploration to improve throughput. Equipped with ANNs for IPC predictions,

strategies for resource allocation [19] and task scheduling [172] can always select decisions that would bring the best predicted IPC. ESP [164] constructs a regression model with elastic-net regularization to predict application interference (i.e., slowdown), which is integrated with schedulers to increase throughput. MetaTune [198] is a meta-learning-based cost model for convolution operations, and when combined with search algorithms, it enables efficient auto-tuning of parameters during compilation. In consideration of rapid design space exploration of the uncore (i.e., memory hierarchies and NoCs), Sangaiah et al. [200] use a regression-based model with restricted cubic splines to estimate CPI, reducing the exploration time by up to four orders of magnitude.

ML-based predictive performance modeling benefits adaptations between performance and power budgets. Leveraging off-line multivariate linear regression to predict IPC and/or power of different architecture configurations, Curtis-Maury et al. [41] maximize performance of OpenMP applications by dynamic concurrency throttling and **dynamic voltage and frequency scaling (DVFS)**; Bailey et al. [12] apply hardware frequency-limiting techniques to select optimal hardware configurations under given power constraints. To effectively apply DVFS towards various optimization goals, the designed strategy can adopt predictions for power consumption by a constrained-posynomial model [102] or job execution time by a linear regression model [140]. To conduct smart power management in a more general manner, LEO [165] employs 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 performance constraints with minimized energy. CALOREE [163] further breaks up the power management task into two abstractions: a learner for performance modeling 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. Since no user-specified parameter except the goal is required, CALOREE is applicable even for non-experts.

3.2.4 Data Center Performance Modeling and Prediction. Data centers have been in widespread use for both traditional enterprise applications and cloud services. Many studies employ ML techniques to predict workload/resource-related metrics to enable elastic resource provision. Common examples include but not limited to using SVR to predict job completion time [253], leveraging the **autoregressive moving average (ARMA)** model [196] or the **autoregressive integrated moving average (ARIMA)** model [27] to forecast incoming workloads, exploiting the hidden Markov modeling to characterize variations in workload patterns [109], and estimating dynamic resource demand of workloads by lightweight statistical learning algorithms [70] or MLP [85]. Jim Gao [65] builds an MLP model to predict power usage effectiveness of data centers, which is extensively tested and validated at Google data centers. Cortez et al. [38] predict **virtual machine (VM)** behaviors (including VM lifetimes, maximum deployment sizes, and workload classes) for a broader set of purposes (e.g., health/resource management and power capping), where the evaluated ML models are random forests and extreme gradient boosting trees.

In addition to workload/resource-related metrics, the availability in data centers or cloud services is also a topic of concern, where one of the key tasks is to predict disk failure in advance. Leveraging **SMART (Self-Monitoring, Analysis and Reporting Technology)** attributes, the disk failure prediction model can be built via various ML techniques, such as different Bayesian methods [75], unsupervised clustering [170], SVM, and MLP [271]. The adoption of **classification and regression trees (CART)** [126], RNNs [248], or gradient boosted regression trees [127] makes it possible to assess health status of drives. While all the aforementioned methods rely on offline training, online random forests [246] can 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 disk failure predictions. To predict partial drive failures (i.e., disk error or sector error), Mahdisoltani et al. [146] explore five ML techniques (CART, random forests, SVM, NN, and logistic regression), among which random forests consistently outperform others. Xu et al. [249] incorporate SMART attributes and system-level signals to train a gradient boosted regression tree, which is an online prediction model that ranks disks according to the degree of error-proneness in the near future.

3.3 Performance Modeling in Chip Design and Design Automation

3.3.1 Analog Circuit Analysis. Analog circuit design is usually a manual process that requires many trial-and-error iterations between pre-layout and post-layout phases. In recent years, the discrepancy between schematic (i.e., pre-layout) performance estimations and post-layout simulation results is further enlarged. On the one hand, the analytical performance estimations from schematics are no longer accurate with device scaling; on the other hand, even though post-layout simulations can provide high-accuracy estimations, they are extremely time-consuming and have become the major bottleneck of design iteration time. To shrink the gap in performance modeling of **integrated circuits (ICs)**, ML techniques are widely applied for fast circuit evaluation.

We discuss the studies based on whether their input features are extracted from pre-layout or post-layout information. ① Given design schematics, parasitics in layouts can be predicted from pre-layout stage, which helps bridge the gap of performance difference between pre-layout and post-layout simulations. ParaGraph [190] builds a GNN model to predict layout-dependent parasitics and physical device parameters. MLParest [210] shows that non-graph based methods (e.g., random forest) also work well for estimating interconnect parasitics, whereas the lack of placement information may cause large variations in predictions. ② Given circuit schematics as well as device information as inputs, it is possible to directly model post-layout performance from pre-layout designs. Alawieh et al. [5] propose a hierarchical method that combines the Bayesian co-learning framework and semi-supervised learning to predict power consumption. The entire circuit schematic is partitioned into multiple blocks to build block-level performance models, upon which circuit-level performance models are built. By combining these two low-dimensional models with a large amount of unlabeled data, pseudo samples can be labeled with almost no cost. Finally, a high-dimensional performance model mapping low-level features to circuit-level metrics is trained with pseudo samples and a small amount of labeled samples, which demonstrates the feasibility of performance modeling with inadequate labeled samples. Several variants of Bayesian-based methods also perform well for estimating post-layout performance, e.g., combining Bayesian regression with SVM to predict circuit performance [179] and using Bayesian DNNs to compare circuit designs [74]. ③ Since post-layout simulations with SPICE-like simulators is time-consuming, ML techniques are applied to quickly assess layout design performance [128]. To make better use of structural information inside layouts, intermediate layout placement results are represented as 3D images to feed a 3D CNN model [138] or encoded as graphs to train a customized GNN model [129], with the goal to predict whether a design specification is satisfied.

3.3.2 High-level Synthesis (HLS). HLS is an automated transformation from behavioral languages (e.g., C/C++/SystemC) to **register-transfer level (RTL)** designs, which significantly expedites the development of hardware designs involving with **field-programmable gate arrays (FPGAs)** or **application-specific integrated circuits (ASICs)**. Since HLS tools usually take considerable time to synthesize each design, it prevents designers from exploring design space sufficiently, which motivates the application of ML models for fast and accurate performance estimation.

In performance estimation of HLS designs, the input features to ML models are extracted from three major sources: HLS directives, IRs from HLS front-ends, and HLS reports. ① Taking the directives in an HLS script as input features, random forest is capable to forecast different design metrics, such as area and effective latency [137] and throughput and logic utilization [159]. To reuse knowledge from previous experiences, a transfer learning approach [119] can transfer knowledge across different applications or synthesis options. ② Taking advantages of IR graphs generated by HLS front-ends, Koeplinger et al. [112] count resource requirements of each node in graphs by using pre-characterized area models, which are then used as inputs to ANNs to predict the LUT routing usage, register duplication, and unavailable LUTs. The exploitation of GNNs makes it possible to automatically predict the mapping from arithmetic operations in IR graphs to different resources on FPGAs [223]. To forecast post-implementation routing congestion, Zhao et al. [265] build a dataset that connects the routing congestion metrics after RTL implementation with operations in IRs, with the goal to train ML models locating highly congested regions in source code. ③ Taking the information that can be directly extracted from HLS reports, Dai et al. [43] try several ML models (linear regression, ANN, and gradient tree boosting) to predict post-implementation resource utilization and timing. Pyramid [148] applies the ensemble learning by stacked regression to accurately estimate the throughput or the throughput-to-area ratio. HL-Pow [136] employs features from both IR graphs and HLS reports to predict the power by a variety of ML models.

The surge of heterogeneous platforms with FPGA/AISC and CPU provides more possibility of hardware/software co-design, motivating cross-platform performance predictions. HLSPredict [176] uses random forest to predict FPGA cycle counts and power consumption based on program counter measurements obtained from CPU execution. While HLSPredict targets the same FPGA platform in training and testing, XPPE [149] considers different FPGA platforms and uses ANNs to predict the speedup of an application on a target FPGA over an ARM processor.

3.3.3 Logic and Physical Synthesis. In digital design, logic synthesis converts RTL designs into optimized gate-level representations; physical synthesis then transforms these design netlists into physical layouts. Since these two stages may take hours or days to generate final bitstreams/layouts, many problems benefit from the power of ML models for fast performance estimation.

In logic synthesis, CNN models [256] or LSTM-based models [258] can be leveraged to forecast the delay and area after applying different synthesis flows on specific designs, where the inputs are synthesis flows represented in either matrices or time series.

In physical synthesis, routing is a sophisticated problem subject to stringent constraints, and EDA tools typically utilize a two-step method: **global routing (GR)** and **detailed routing (DR)**. GR tool allocates routing resource coarsely and provides routing plans to guide DR tools to complete the entire routing. In general, routing congestion can be figured out during or after GR; the routability of a design is confirmed after DR and **design rule checking (DRC)**. Endeavors have been made to predict routability from early layout stages to avoid excessive iterations back and forth between placement and routing.

In ASICs, some investigations predict routability by estimating the number of **design rule violations (DRVs)**. Taking GR results as inputs, Li et al. [125] explore several ML models (linear regression, ANN, and decision tree) to predict the number of DRVs, final hold slack, power, and area. Qi et al. [184] rely on placement data and congestion maps from GR as input features and use a nonparametric regression technique, **multivariate adaptive regression splines (MARS)** [62], to predict the utilization of routing resource and the number of DRVs. By merely leveraging placement information, it is possible to predict routability by MARS and SVM [28] or to detect DR short violations by an MLP [216]. When representing placement information as images, **fully**

convolutional networks (FCNs) are capable to predict locations of DRC hotspots by considering GR information as inputs [247] or to forecast GR congestion maps by formulating the prediction task as a pixel-wise binary classification using placement data [29]. J-Net [131] is a customized FCN model and takes both high-resolution pin patterns and low-resolution layout information from the placement stage as features to output a 2D array that indicates if the tile corresponding to each entry is a DRC hotspot.

In FPGAs, routing congestion maps can be directly estimated by linear regression [145] using feature vectors coming from pin counts and wirelength per area of SLICEs. By constructing the routing congestion prediction as an image translation problem, a conditional generative adversarial network (GAN) [6, 257] is able to take post-placement images as inputs to predict congestion heat maps.

4 ML AS DESIGN METHODOLOGY

This section introduces studies that directly employ ML techniques as the design methodology for computer architecture/systems. Computer architecture and systems have been becoming increasingly complicated, making it expensive and inefficient for human efforts to 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, and so on. With strong learning and generalization capabilities, ML-based techniques are naturally suitable to resolve these considerations, which can adjust their policies during system designs according to long-term planning and dynamic workload behaviors. As many problems in architecture/system design can be formulated as combinatorial optimization or sequential decision-making problems, RL is broadly explored and exploited. Tables 4 and 5 recapitulate the studies that apply ML techniques as the design methodology for computer architecture/system and design automation, respectively, in terms of target tasks and adopted ML techniques.

4.1 Memory System Design

The "memory wall" has been a performance bottleneck in von Neumann architectures, where computation is orders of magnitude faster than 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 designs that are based on heuristics or intuitions may not catch up with the demand of the ever-growing workloads, leading to sharp degradation in performance. As such, many studies resort to ML-based techniques to design smart and intelligent memory systems.

4.1.1 Cache. The conspicuous disparity in latency and bandwidth between CPUs and memory systems motivates investigations for efficient cache management. There are two major types of studies on cache optimization: improving cache replacement policies and designing intelligent prefetching policies. ① To develop cache replacement policies, perceptron learning is employed to predict whether to bypass or reuse a referenced block in the **last-level cache (LLC)** [96, 219]. Instead of using perceptrons, Beckmann et al. [16] model the cache replacement problem as a Markov decision process and replace lines according to the difference between their expected hits and the average hits. Shi et al. [207] train an attention-based LSTM model offline to extract insights from history program counters, which are then used to build an online SVM-based hardware predictor to serve as the cache replacement policy. ② To devise intelligent prefetchers, Wang et al. [230] 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.

Table 4. Summary of Applying ML Techniques as the Design Methodology for Computer Architecture/Systems

Domain	Task	Technique
Memory System Design (Section 4.1)	Cache replacement policy	Perceptron learning [96, 219], Markov decision process [16], LSTM and SVM [207]
	Cache prefetching policy	Perceptron learning [17, 230], contextual bandit [183], LSTM [24, 77, 208, 261]
	Memory controller policy	Q-learning [84, 153, 169]
	Garbage collection	Q-learning [104, 105]
Branch Prediction (Section 4.2)	Branch direction	MLP [26], piecewise linear regression [92], perceptron [67, 91, 93–95, 213], CNN [218]
NoC (Section 4.3)	Link management	ANN [192, 201]
	DVFS for routers	Q-learning [60, 266]
	Routing	Q-learning [23, 54, 59, 118, 147]
	Arbitration policy	DQN [254, 255]
	Adjusting injection rates	Q-learning [46], ANN [229]
	Selection of fault-tolerant modes	Q-learning [233, 234]
	Link placement in 3D NoCs	STAGE algorithm [44, 45, 97]
	Loop placement in routerless NoCs	Advantage actor-critic with MCTS [134]
Power Management (Section 4.4.1)	DVFS and thread packing	Multinomial logistic regression [36]
	DVFS and power gating	MLP [187]
	DVFS, socket allocation, and use of HyperThreads	Extra trees/gradient boosting/KNN/MLP/SVM [82]
	DVFS for CPU cores/uncore/through-silicon interposers	Propositional rule [1], ANN [238], Q-learning [182]
	DVFS for CPU-GPU heterogeneous platforms	Weighted majority algorithm [144]
	DVFS for multi-/many-core systems	Q-learning [11], semi-supervised RL [103], hierarchical Q-learning [32, 33, 178]
Resource Management & Task Allocation (Section 4.4.2)	Tuning architecture configurations	Maximum likelihood [53], statistical machine learning [20, 64]
	Dynamic cache partitioning	Enforced subpopulations [69], Q-learning [89]
	Task allocation in many-core systems	Q-learning [142], DDPG [241]
	Workflow management	SVM and random forest [56]
	Hardware resource assignment	REINFORCE [106], Bayesian optimization [98]
	Device placement	REINFORCE [160, 162], policy gradient [2], PPO [66, 270]
Scheduling (Section 4.4.3)	Scheduling jobs in single-core processors	Q-learning [236]
	Scheduling jobs in multi-processor systems	Value-based RL [58, 226]
Data Center Management (Section 4.5)	Assignment of servers to applications	Value-based RL [220, 221]
	Content allocation in CDNs	Fuzzy RL [227]
	Placement of virtual machines onto physical machines	PPO [13]
	Traffic optimization	Policy gradient and DDPG [30]
	Scheduling jobs with complex dependency	REINFORCE [151]
	Straggler diagnosis	Statistical ML [267]
	Data-center-level caching policy	Decision tree [232], LSTM [171], gradient boosting [211], DDPG [242]
	Bitrate selection for video chunks	A3C [150, 252]
	Scheduling video workloads in hybrid CPU-GPU clusters	DQN [263]
Code Generation (Section 4.6.1)	Code completion	N-gram model and RNN [188]
	Code generation	LSTM [40]
	Program translation	Tree-to-tree encoder-decoder [31, 63], seq2seq [111], transformer [120]
Compiler (Section 4.6.2)	Instruction scheduling	Temporal difference [154], projective reparameterization [87]
	Improving compiler heuristics	NEAT [37], LSTM [39]
	Ordering of optimizations	NEAT [114]
	Automatic vectorization	Imitation learning [158]
	Program transformation for approximate computing	MLP [55, 251]
	Compilation for DNN workloads	PPO [4], policy gradient [107]

[17] integrate a perceptron-based prefetching filter with conventional prefetchers, increasing the coverage of prefetches without hurting accuracy. Instead of the commonly used spatio-temporal locality, a context-based memory prefetcher [183] leverages the semantic locality that characterizes access correlations inherent to program semantics and data structures, which is approximated by a contextual bandit model in RL. Interpreting semantics in memory access patterns is 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 [261] or a classification problem [77]. Even with better performance, especially for long access sequences and noise traces, LSTM-based prefetchers suffer from long warm-up and prediction latency, and considerable storage overheads. The discussion of how hyperparameters impact LSTM-based prefetchers' performance [24] highlights that the lookback size (i.e., memory access history window) and the LSTM model size strongly affect prefetchers' learning ability under different noise levels or workload patterns. To accommodate the large memory space, Shi et al. [208] introduce a hierarchical sequence model to decouple predictions of pages and offsets by using two separate attention-based LSTM layers, whereas the corresponding hardware implementation is impractical for actual processors.

Table 5. Summary of Applying ML Techniques as the Design Methodology for Design Automation

Domain		Task	Technique
Analog Design (Section 4.7.1)	Circuit Level	Generating circuit topology	RNN and hypernetwork [195]
	Device Level	Device sizing	Actor critic [205, 231], ANN [194]
	Physical Level	Routing	VAE [272]
		Optimizing layout configurations	Multi-objective Bayesian optimization [139]
Digital Design (Section 4.7.2)	HLS	Optimizing loop unrolling pragma	Random forest [259]
		Optimizing placement of multiple pragmas	Bayesian optimization [155]
		Optimizing resource pragma	Actor-critic and GNN [243]
	Logic Synthesis	Selecting proper optimizers	MLP [173]
		Logic optimization	Policy gradient [73], actor-critic [81]
		Determining the maximum error of each node	Q-learning [180]
	Physical Synthesis	Optimizing flip-flop placement in clock networks	K-means clustering [240]
		Optimizing clock tree synthesis	Conditional GAN [143]
		Optimizing memory cell placement	PPO [161]
		Optimizing standard cell placement	Cast as an NN training problem [135]
		Fixing design rule violations	PPO [189]

4.1.2 Memory Controller. Smart memory controllers can significantly improve memory bandwidth utilization. Aiming at a self-optimizing memory controller adaptive to dynamically changing workloads, it can be modeled as an RL agent that always selects legal DRAM commands with the highest expected long-term performance benefits (i.e., Q-values) [84, 153]. To allow optimizations toward various objectives, this memory controller is then improved in two major aspects [169]. 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). Second, a multi-factor method that considers the first-order attribute interactions is employed to select proper 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. A variety of work targets different parts of the memory system. Margaritov et al. [152] accelerate virtual address translation through learned index structures [113]. The results are encouraging in terms of the accuracy, which reaches almost 100% for all tested virtual addresses; yet this method has unacceptably long inference latency, leaving practical hardware implementation as the future work. Wang et al. [235] reduce data movement energy in interconnects by exploiting asymmetric transmission costs of different bits, where data blocks to be transmitted are dynamically grouped by K-majority clustering to derive energy-efficient expressions for transmission. In terms of garbage collection in NAND flash, Kang et al. [104] 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, where decisions are made by table-based Q-learning. Their following work [105] considers more fine-grained states and introduces a Q-table cache to manage key states among an 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 trained with static features from program corpus and control flow graphs, an MLP is used to predict the direction of a branch at compile time [26]. Later, a dynamic branch predictor uses a perceptron-based method [95]. It hashes the branch address to select the proper perceptron and computes the dot product accordingly to decide whether to

take this branch, which shows 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 [91]. To attain high accuracy in non-linearly separable branches, the perceptron-based prediction is generalized as piecewise linear branch prediction [92]. In addition to the path history, multiple types of features from different organizations of branch histories can be leveraged to enhance the overall performance [94]. When considering practical hardware implementation of branch predictors, SNAP [213] leverages 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 [93] equips several new techniques, such as the use of global and per-branch history, trainable scaling coefficients, dynamic training thresholds, and so on.

Rather than making binary decisions of whether to take a certain branch, it is possible to directly predict the target address of an indirect branch at the bit level via perceptron-based predictors [67]. While high accuracy is achieved by current perceptron/MLP-based predictors, Tarsa et al. [218] notice that a small amount of static branch instructions are systematically mispredicted, referred to as **hard-to-predict branches (H2Ps)**. Consequently, they propose a CNN helper predictor for pattern matching of history branches, 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 plays a gradually crucial role, since it is responsible for inter-core communication and data movement between cores and memory hierarchies. Several problems attracting attention are as follows. First, communication energy scales slower than computation energy [22], implying necessity to improve power efficiency of NoCs. 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. Fourth, some non-conventional NoC architectures might bring promising potentials in the future, whereas they usually come with large design spaces and complex design constraints, which is nearly impossible for manual optimization. Among all aforementioned fields, 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. [201] use multiple ANNs for dynamic link management. Each ANN is responsible for one region of the NoC and dynamically computes a threshold for every time interval to turn on/off links given the link utilization of each region. Despite significant power savings with low hardware overheads, this approach causes long latency in routing. 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 [60, 266] employ per-router based Q-learning agents, which are 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. ① As routing problems can be formulated as sequential decision-making processes, several studies apply Q-learning-based approaches, namely, the Q-routing algorithm [23], which uses local estimation of delivery time to minimize total packets delivery time, capable to handle irregular network topologies and keep a higher network load than the conventional shortest path routing. Q-routing is then extended to several other scenarios, such as combining with dual RL to improve learning speed and routing performance [118], resolving packets routing in dynamic NoCs whose network structures/topologies are dynamically changing during runtime [147], handling irregular faults in bufferless NoCs by the reconfigurable fault-tolerant Q-routing [59], and enhancing the capability to reroute messages around congested regions by the congestion-aware non-minimal Q-routing [54]. In addition to routing problems, deep Q-network is also promising for NoC arbitration policies [254, 255], where the agent/arbiter grants a certain output port to the input buffer with the largest Q-value. Even displaying some improvements in latency and throughput, the direct hardware implementation is impractical due to the complexity of deep Q-networks, and thus insights are distilled to derive a relatively simple circuitry implementation. ② With the goal to control congestion in NoCs, the SCEPTER NoC architecture [46], 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. Each node in the network independently selects whether to increase, decrease, or retain the throttle rate according to their Q-values, which conspicuously improves bandwidth allocation fairness and network throughput. Wang et al. [229] design an ANN-based admission controller to determine the appropriate injection rate and the 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. [233] employ per-router-based Q-learning agents to independently select one of four fault-tolerant modes, which can minimize the end-to-end packet latency and power consumption. These agents are pre-trained and then fine-tuned during runtime. In their following work [234], these error-correction modes are extended and combined with various multi-function adaptive channel configurations, retransmission settings, and power management strategies, significantly improving latency, 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 simultaneously optimize copious design knobs in NoCs. One attempt to automated NoC design 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 and evaluated 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 3D NoC designs, where the STAGE algorithm is applied to optimize vertical and planar placement of communication links in small-world network-based 3D NoCs [44, 45]. The STAGE algorithm repeatedly alternates between two stages: the base search that tries to find the local optima based on the learned evaluation function, 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 [97], which jointly considers 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, a deep RL framework that exploits Monte-Carlo tree search for efficient design space exploration

is developed to optimize loop placements [134], 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.

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.

In consideration of power management for different parts of systems, PACSL [1] uses the propositional rule to adjust **dynamic voltage scaling (DVS)** for CPU cores and on-chip L2 cache, which achieves an improvement in the energy-delay product by 22% on average (up to 46%) over independently applying DVS for each part. Won et al. [238] coordinate an ANN controller with a proportional integral for uncore DVFS. The ANN controller can be either pre-trained offline by a prepared dataset or trained online by bootstrapped learning. Manoj et al. [182] 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 communication power and bit error rate.

From the system level, DVFS is one of the most prevalent techniques. Pack & Cap [36] build a multinomial logistic regression classifier that 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 [144] focuses on heterogeneous systems with CPUs and GPUs, and applies the weighted majority algorithm to scale frequency levels for both GPU cores and memory in a coordinated manner. CHARSTAR [187] targets joint optimization of power gating and DVFS within a single core, where frequencies and configurations are dynamically selected by a lightweight offline-trained MLP predictor. To minimize energy consumption, Imes et al. [82] use ML-based classifiers (e.g., extra trees, gradient boosting, KNN, MLP, and SVM) 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. Bai et al. [11] consider the loss caused by on-chip regulator efficiency during DVFS and try to minimize energy consumption under a parameterized performance constraint. The online control policy is implemented by a table-based Q-learning, which is portable across platforms without accurate modeling of a specific system.

A series of studies leverages RL for dynamic power management in multi-/many-core systems. As systems scale up, these RL-based methods often suffer from state space explosion, and two types of methods are introduced to resolve the scalability issue: ① By combining RL with supervised learning, a semi-supervised RL-based approach [103] 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. ② The exploitation of hierarchical Q-learning reduces the time complexity to $O(n \lg n)$, where n denotes the number of cores. Pan et al. [178] introduce multi-level Q-learning to select target power modes, where Q-values are approximated by a generalized radial basis function. Table-based distributed Q-learning also performs well for DVFS [32], and there is one variant [33] aware of the priorities of different applications.

Some energy management policies target specific applications or platforms. JouleGuard [79] is a runtime control system coordinating approximate computing applications with system resource under energy budgets. It uses a multi-arm bandit approach to identifying the most energy

efficient system configuration, upon which application configurations are determined to maximize compute accuracy within energy budgets. Targeting Intel SkyLake processors, a post-silicon CPU customization applies various ML models for dynamically clock-gating unused resource [217].

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 resource. This rapidly evolving landscape is further complicated by various workloads with specific requirements or targets. To keep the pace, one cure is to develop more efficient and automated methods for resource management and task allocation, where ML-based techniques are excelled to explore large design spaces, simultaneously optimize multiple objectives, and preserve better scalablility and portability after being carefully designed.

For a single-core processor, a regularized maximum likelihood approach [53] predicts the best hardware micro-architectural configuration for each phase of a program, based on runtime hardware counters. For multi-core processors, a **statistical machine learning (SML)**-based method [64] can quickly find configurations that simultaneously optimize running time and energy efficiency. Since this method is agnostic to application and micro-architecture domain knowledge, it is a portable alternative to human expert optimization. SML can also be applied as a holistic method to design self-evolving systems that optimize performance hierarchically across circuit, platform, and application levels [20]. In addition to tuning architectural configurations, dynamic on-chip resource management is crucial for multi-core processors, where one example is dynamic cache partitioning. In response to changing workload demands, an RNN evolved by the enforced subpopulations algorithm [69] is introduced to partition L2 cache dynamically. When integrating dynamic partitioning of LLC with DVFS on cores and uncore, a co-optimization method using table-based Q-learning achieves much lower energy-delay products than any of the techniques applied individually [89].

To guarantee efficient and reliable execution in many-core systems, task allocation should consider several aspects, such as heat and communication issues. Targeting the heat interaction of processor cores and NoC routers, Lu et al. [142] apply Q-learning to assign tasks to 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 [241] leverages **deep deterministic policy gradient (DDPG)** [132] to map computation onto physical cores, able to work in a manner agnostic to domain-specific information.

Some studies pay attention to workflow management and general hardware resource assignment. SmartFlux [56] focuses on the workflow of data-intensive and continuous processing. It intelligently guides asynchronous triggering of processing steps with the help of predictions made by multiple ML models (e.g., SVM, random forest), which indicate whether to execute certain steps and decide corresponding configurations upon each wave of data. Given target DNN models, deployment scenarios, platform constraints, and optimization objectives (latency/energy), ConfuciuX [106] 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 REINFORCE [214] performs a global coarse-grained search followed by a genetic algorithm for fine-grained tuning. Apollo [98] is a general architecture exploration framework for sample-efficient accelerator designs, which leverages ML-based black-box optimization techniques (e.g., Bayesian optimization) to optimize accelerator configurations to satisfy user-specified design constraints.

In heterogeneous systems with CPUs and GPUs, device placement refers to the process of mapping nodes in computational graphs of neural networks onto proper hardware devices. Initially, computational operations are grouped manually, and assigned to devices by REINFORCE, which

employs a sequence-to-sequence RNN model as the parameterized policy [162]. Later, a hierarchical end-to-end model makes this manual grouping process automatic [160]. The training speed is further improved by introduction of **proximal policy optimization (PPO)** [66]. Despite great advance brought by the above approaches, they are not transferable and a new policy should be trained from scratch specifically for each new computational graph. By encoding structure of computational graphs with static graph embeddings [2] or learnable graph embeddings [270], the trained placement policy exhibits great generalizability to unseen neural networks.

4.4.3 Scheduling. In classical real-time scheduling problems, the key task is to decide the order, 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 to 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 core, Whiteson and Stone [236] propose an adaptive scheduling policy that exploits Q-routing, where the scheduler utilizes the router's Q-table to assess a job's priority and decides jobs' ordering accordingly to maximize the overall utility. In multi-core systems, Fedorova et al. [58] present a blueprint for a self-tuning scheduling algorithm based on the value-based temporal-difference method in RL, aiming to maximize a cost function that is an arbitrary weighted sum of metrics of interest. This algorithm is then improved to be a general method for online scheduling of parallel jobs [226], where the value functions are approximated by a parameterized fuzzy rulebase. This scheduling policy always selects to execute jobs with the maximum value functions in the job queue, which possibly preempts currently running jobs and squeezes some jobs into fewer CPUs than they ideally require, with the goal of achieving optimized long-term utility.

4.5 Data Center Management

With the rapid scale expansion 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, i.e., to dynamically assign different numbers of servers to multiple applications. This problem can be modeled as an RL problem with service-level utility functions as rewards: the arbiter will select a joint action that would bring the maximum total return after consulting local value functions estimated via either table-based methods [221] or function approximation [220]. To better model interactions among multiple agents, a multi-agent coordination algorithm with fuzzy RL [227] 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. A recent innovation [13] pays attention to the placement of virtual machines onto physical machines to minimize the peak-to-average ratio of resource usage across physical machines, where PPO and hindsight imitation learning are evaluated.

To improve data center performance and **quality of experience (QoE)** for users, ML-based techniques have been explored in a few directions. ① It is important to efficiently schedule jobs and effectively diagnose stragglers within jobs. Aiming at traffic optimization (e.g., flow scheduling, load balancing) in data centers, Chen et al. [30] develop a two-level RL system: peripheral systems, which are trained by DDPG, reside on end-hosts and locally make instant traffic optimization decisions for short flows; the central system, which is trained by policy gradient, aggregates global traffic information, guides behaviors of peripheral systems, and makes traffic optimization

decisions for long flows. Decima [151] exploits GNNs to represent cluster information and dependency among job stages, so the RL-based scheduler can automatically learn workload-specific scheduling policies to schedule data processing jobs with complex dependency. Hound [267] combines statistical ML with meta-learning to diagnose causes of stragglers at data-center-scale jobs. ② It is essential to deploy an intelligent data-center-level cache. DeepCache [171] employs an LSTM encoder-decoder model to predict future content popularity, which can be combined with existing cache policies to make smarter decisions. Song et al. [211] apply gradient boosting machines to mimic a relaxed Belady algorithm that evicts an object whose next request is beyond a reuse distance threshold but not necessarily the farthest in the future. Phoebe [242] is an online cache replacement framework leveraging DDPG to predict priorities of objects and to conduct eviction accordingly. Considering non-history based features, Wang et al. [232] build a decision tree to predict whether the requested file will be accessed only once in the future. These one-time-access files will be directly sent to users without getting into cache to avoid cache pollution. ③ From the workload perspective, video workloads on CDNs or clusters are prevalent but their optimization is quite challenging: first, network conditions fluctuate over time and a variety of QoE goals should be balanced simultaneously; second, only coarse decisions are available and 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, adaptive bitrate algorithms have been recognized as the primary tool used by content providers, which are executed on client-side video players and dynamically choose a bitrate for each video chunk based on underlying network conditions. Pensieve [150, 252] applies asynchronous advantage actor-critic [166] to select proper bitrate for future video chunks based on resulting performance from past decisions. When considering large-scale video workloads in hybrid CPU-GPU clusters, performance degradation often comes from uncertainty and variability of workloads and unbalanced use of heterogeneous resources. To accommodate this, Zhang et al. [263] use two deep Q-networks to build 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. This scheme enables the scheduling model to adjust policies according to runtime status of cluster environments, characteristics of video tasks, and dependency among video tasks.

4.6 Code Generation and Compiler

4.6.1 Code Generation. Due to the similarities in syntax and semantics between programming languages and natural languages, the problem of code generation or translation is often modeled as an NLP problem or a **neural machine translation (NMT)** problem. Here, we would like to bring up a brief discussion. For more reference, a comprehensive survey [7] detailedly contrasts programming languages against natural languages and discusses how these similarities and differences drive the design and application of different ML models in code.

Targeting code completion, several statistical language models (N-gram model, RNN, and a combination of these two) [188] are explored to select sentences that have the highest probability and satisfy constraints to fill up partial programs with holes. As for code generation, CLgen [40] trains LSTM models by a corpus of hand-written code to learn semantics and structures of OpenCL programs, and generates human-like programs via iteratively sampling from the learned model.

Targeting program translation, NMT-based techniques are widely applied to migrate code from one language to another. For example, a tree-to-tree model with the encoder-decoder structure effectively translates programs from Java to C# [31]; the **sequence-to-sequence (seq2seq)** model can translate from CUDA to OpenCL [111]. Rather than translating between high-level programming languages, Coda [63] translates binary executables to the corresponding high-level code,

which employs a tree-to-tree encoder-decoder structure for code sketch generation and an ensembled RNN-based error predictor for iterative error correction on the generated code. Notably, these supervised NMT-based techniques may confront several issues: difficulty to generalize to programs longer than training ones, limited size of vocabulary sets, and scarcity of aligned input-output data. Fully counting on unsupervised machine translation, TransCoder [120] adopts a transformer architecture and uses monolingual source code to translate among C++, Java, and Python.

4.6.2 Compiler. The complexity of compilers grows with the complexity of computer architectures and workloads. ML-based techniques can optimize compilers from many perspectives, such as instruction scheduling, compiler heuristics, the order to apply optimizations, hot path identification, auto-vectorization, and compilation for specific applications. ① For instruction scheduling, the preference function of one scheduling over another can be computed by the temporal difference algorithm in RL [154]. Regarding scheduling under highly constrained code optimization, the projective reparameterization [87] enables automatic instruction scheduling under constraints of data-dependent partial orders over instructions. ② For improving compiler heuristics, **NeuroEvolution of Augmenting Topologies (NEAT)** [37] improves instruction placement heuristics by tuning placement cost functions. To avoid manual feature engineering, an LSTM-based model [39] automatically learns compiler heuristics from raw code, which constructs proper embeddings of programs and simultaneously learns the optimization process. ③ For choosing the appropriate order to apply different optimizations, NEAT [114] can automatically generate beneficial optimization orderings for each method in a program. ④ For path profiling, CrystalBall [260] uses an LSTM model to statically identify hot paths, the sequences of instructions that are frequently executed. As CrystalBall only relies on IRs, it avoids manual feature crafting and is independent of language or platform. ⑤ For automatic vectorization, Mendis et al. [158] leverage imitation learning to mimic optimal solutions provided by superword-level-parallelism-based vectorization [156]. ⑥ For compilation of specific applications, there are studies improving compilation for approximate computing or DNN applications. Considering compilation for approximate computing, Esmaeilzadeh et al. [55] propose a program transformation method, which trains MLPs to mimic regions of approximable code and eventually replaces the original code with trained MLPs. The following work [251] extends this algorithmic transformation to GPUs. Considering compilation for DNNs, RELEASE [4] utilizes PPO to search optimal compilation configurations for DNNs. EGRL [107] optimizes memory placement of DNN tensors during compilation, which combines GNNs, RL, and evolutionary search to figure out optimal mapping onto different on-board memory components (i.e., SRAM, LLC, and DRAM).

4.7 Chip Design and Design Automation

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

4.7.1 Analog Design. Compared with the highly automated digital design counterpart, analog design usually demands many manual efforts and domain expertise. First, analog circuits have large design spaces to search proper topology and device sizes. Second, there is an absence of a general framework to optimize or evaluate analog designs, and design specifications often vary case-by-case. Recently, ML techniques have been introduced to expedite analog design automation. We discuss these studies following the top-down flow of analog design: in the circuit level a proper circuit topology is selected to satisfy system specifications; then, in the device level, device sizes are optimized subject to various objectives. These two steps constitute pre-layout designs. After circuit schematics are carefully designed, analog layouts in the physical level are generated.

① In the circuit level, there is an attempt towards automatic circuit generation currently targeting two-port linear analog circuits [195]. The design specifications are encoded by a hypernetwork [72] to generate weights for an RNN model, which is trained to select circuit components and their configurations. ② In the device level, the combination of RL and GNNs enables automatic transistor sizing [231], which is able to generalize across different circuit topologies or different technology nodes. AutoCkt [205] introduces transfer learning techniques into deep RL for automatic sizing, achieving 40× speedup over a traditional genetic algorithm. Rosa et al. [194] provide comprehensive discussions of how to address automatic sizing and layout of analog ICs via deep learning and ANNs. ③ In the physical level, GeniusRoute [272] automates analog routing through the guidance learned by a generative neural network. The analog placements and routing are represented as images to pass through a **variational autoencoder (VAE)** [71] to learn routing likelihoods of each region. GeniusRoute achieves competitive performance to manual layouts and is capable to generalize to circuits of different functionality. Liu et al. [139] apply multi-objective Bayesian optimization to optimize combinations of net weighting parameters, which could significantly change floor plans and placement solutions to improve analog layouts of building block circuits.

4.7.2 Digital Design. For the studies applying ML techniques to directly optimize digital designs, we organize them following a top-down flow, i.e., HLS, logic synthesis, and physical synthesis.

The design space exploration in HLS designs usually relates to properly assigning directives (pragmas) in high-level source code, since directives significantly impact the quality of HLS designs by controlling parallelism, scheduling, and resource usage. The optimization goal is often to find Pareto solutions between different objectives or to satisfy pre-defined constraints. With IR analysis, the employment of a random forest is able to select suitable loop unrolling factors to optimize a weighted sum of execution latency and resource usage [259]. Prospector [155] uses Bayesian optimization to optimize placement of directives (loop unrolling/pipelining, array partitioning, function inlining, and allocation), aiming to find Pareto solutions between execution latency and resource utilization in FPGAs. IronMan [243] targets Pareto solutions between different resources while keeping the latency unchanged. It combines GNNs with RL to conduct a finer-grained design exploration in the operation level, pursuing optimal resource allocation strategies by optimizing assignments of the resource pragma.

In logic synthesis, RTL-designs or logic networks are represented by **directed acyclic graphs (DAGs)**. The goal is to optimize logic networks subject to certain constraints. LSOracle [173] employs an MLP to automatically decide which one of the two optimizers should be applied on different parts of circuits. The logic optimization can be formulated as an RL problem solved by the policy gradient [73] or the advantage actor-critic [81]: the state is the current status of a design; the action is a transformation between two DAGs with equivalent I/O behaviors; the optimization objective is to minimize area or delay of designs. Q-ALS [180] aims at approximate logic synthesis and embeds a Q-learning agent to determine the maximum tolerable error of each node in a DAG, such that the total error rates at primary outputs are bounded by pre-specified constraints.

In physical synthesis, placement optimization is a popular topic. ① To optimize flip-flop placement in clock networks, Wu et al. [240] apply a modified K-means clustering to group post-placement flip-flops and relocate these clusters by reducing the distance between flip-flops and their drivers while minimizing disruption of original placement results. To optimize **clock tree synthesis (CTS)**, Lu et al. [143] train a regression model that takes pre-CTS placement images and CTS configurations as inputs to predict post-CTS metrics (clock power, clock wirelength, and maximum skew), which is used as the supervisor to guide the training of a conditional GAN, such that the well-trained generator can recommend CTS configurations leading to optimized clock

trees. ② Aiming at cell placement, a deep RL approach [161] is introduced to place macros (memory cells), after which standard cells are placed by a force-directed method. This method is able to generalize to unseen netlists and outperforms RePlAce [34] yet several times slower. DREAMPlace [135] casts the analytical standard cell placement optimization into a neural network training problem, achieving over 30× speedup without quality degradation compared to RePlAce. NVCell [189] is an automated layout generator for standard cells, which employs RL to fix DRVs after placement and routing. ③ ML-based techniques also demonstrate their versatility in other many design automation tasks, such as post-silicon variation extraction by sparse Bayesian learning and post-silicon timing tuning to mitigate the effects caused by process variation [274].

5 DISCUSSION AND POTENTIAL DIRECTIONS

In this section, we discuss limitations and potentials of ML techniques for computer architecture and systems, which span the entire development and deployment stack that involves data, algorithms, implementation, and targets. We also envision that the application of ML techniques could be the propulsive force for *hardware agile development*.

5.1 Bridging Data Gaps

Data are the backbone to ML, however, perfect datasets are sometimes non-available or prohibitively expensive to obtain in computer architecture and system domain. Here, we would like to scrutinize two points: the gap between small data and big data, and non-perfect data. ① In some EDA problems, such as placement and routing in physical synthesis, the simulation or evaluation is extremely expensive [250], leading to data scarcity. As ML models usually require enough data to learn underlying 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 in small parameter space [108]; active learning [206], which significantly improves sample efficiency, may also be a cure to this problem. From the data side, generative methods can be used to generate synthetic data [49], mitigating data scarcity. ② Regarding non-perfect data, even if some EDA tools produce a lot of data, they are not always properly labeled nor presented in the form suitable to ML models. In the absence of perfectly labeled training data, possible alternatives are to use unsupervised learning, self-supervised learning [78], or to combine supervised with unsupervised techniques [5]. Meanwhile, RL could be a workaround where training data can be generated on-the-fly.

5.2 Developing Algorithms

Despite the current achieved accomplishments, we are still expecting novel ML algorithms or schemes to further improve system modeling and optimization, with respect to scalability, domain knowledge interpretability, and so on.

New ML Schemes. Classical analytic-based methods usually adopt a bottom-up or top-down procedure, encouraging ML-based techniques to distill hierarchical structures of systems/architecture. One example is hierarchical RL [115] that has flexible goal specifications and learns 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 [264]. 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 [78], 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, RL can be combined with genetic algorithms for hardware resource assignment [106].

Scalability. The system scaling-up poses challenges on 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, 178]. 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, 205, 231] are discussed in Section 4.7.

Domain Knowledge and Interpretability. Making better use of domain knowledge unveils possibilities to choose more proper models for different system problems and provide more intuitions or explanations of why and how these models work. By making analogy of semantics between memory access patterns/program languages and natural languages, the prefetching or code generation problems can be modeled as NLP problems, as discussed in Section 4.1.1 and Section 4.6.1. By making analogy of graphical representations in many EDA problems, where data are intrinsically presented as graphs (e.g., circuits, logic netlists, or IRs), GNNs are expected to be powerful in these fields [108]. Several examples are provided in Section 3.3 and Section 4.7.

5.3 Improving Implementations and Deployments

To fully benefit from ML-based methods, we need to consider practical implementations, appropriate selection of deployment scenarios, and post-deployment model maintenance.

Better Implementations. To enable practical implementations of ML-based techniques, improvement can be made from either the model side or software/hardware co-design [215]. From the model level, network pruning and model compression reduce the number of operations and model size [76]; weight quantization improves computation efficiency by reducing the precision of operations/operands [86]. 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 design constraints under different scenarios. Generally, existing work falls into two categories ① ML-based techniques are deployed 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. To take one more step, if the online training/learning is further desired, then 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 [116] and caching [117]. These models enable smooth interpolation between the best possible online and offline training algorithms. ② ML-based techniques are applied offline, which often refers to architectural design space exploration. Such problems leverage ML-based techniques to guide system implementation, and once the designing phase is completed, ML models will not be invoked again. Thus, the offline applications can tolerate relatively higher overheads.

Model Maintenance. In the case of offline training and online deployment, ML models employed for computer architecture domain, as in other scenarios, require regular maintenance and updating to meet performance expectations, since workload variations over time and hardware aging often cause data drift or concept drift [222]. To proactively circumvent performance degradation of ML models, some measures could be taken during post-deployment periods. ① ML models can be retrained either at a regular interval or when key performance indicators are below certain thresholds. Retraining models regularly, regardless of their performance, is a more direct way, but it requires a clear understanding of how frequently a model should be updated under its

own scenario. The model performance will decline if retraining intervals are too spaced out in the interim. Monitoring key performance indicators relies on a comprehensive panel of measurements that explicitly demonstrate model drift, whereas this may introduce additional hardware/software overhead and incorrect selection of measurements often defeats the intention of this method. ② During the retraining of ML models, there is often a tradeoff between newly collected data and previous data. Properly assigning importance of input data would improve retraining efficacy [25].

5.4 Supporting Non-homogeneous Tasks

ML-based techniques are supposed to be applicable in both current architectures and emerging systems, leading to long-term advancement in computer architecture and systems.

Non-homogeneous Components. Design and development for computer architectures are often based upon earlier-generation architectures of similar purpose, but commonly rely on next-generation hardware components that were not present in earlier generations. Examples include employment of new device nodes with technology scaling and replacement of conventional constituents in memory systems with NVM- or PIM-based components. In addition to the heterogeneity of components from different generations, one architecture or system usually consists of both standard parts from library and specialized/customized hardware components. This provides the motivation that ML-assisted architectures/systems should have the flexibility to transfer among different-generation components and to support standard and specialized parts simultaneously.

Non-homogeneous Applications. In computer architecture and system design, some issues are universal, while others may arise with the advent of new architecture/systems and new workloads. ① For evergreen design areas, several examples include caching in hardware/software/data centers (Section 4.1.1 and Section 4.5), 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), and so on. ② For problems aroused from new systems/workloads, transfer learning and meta-learning [174, 225] could be helpful in either exploring new heuristics or directly deriving design methodology. For example, combining meta-learning with RL [61] allows training a "meta" agent that is designed to adapt to a specific workload with only a few observations.

5.5 Facilitating General Tool Design and Hardware Agile Development

Even though ML-based modeling significantly reduces the evaluation cost during design iteration, making great strides towards the landing of hardware agile development, there is still a long way to go in the ML-based design methodology prospective. One ultimate goal might be the fully automated design, which should entangle two core capabilities: holistic optimization in system-wise, and easy migration across different systems, to enable rapid and agile hardware design.

Holistic Optimization. Fueled by recent advancements, ML techniques have been increasingly explored and exploited in computer system design and optimization [47]. The target problems that await further endeavors could be multi-objective optimizations under highly constrained situations or optimizing several components in a system simultaneously. We envisage an ML-based system-wise and holistic framework with a panoramic vision: it should be able to leverage information from different levels of systems in synergy so it can thoroughly characterize system behaviors as well as their intrinsically hierarchical abstractions; it should also be able to make decisions in different granularity so it can control and improve systems precisely and comprehensively.

Portable, Rapid, and Agile. Striving for portable, rapid, and agile hardware design, there are two potential directions. ① The well-designed interfaces between systems/architectures and ML-based techniques would facilitate the portability across different platforms, since ML models can perform well without explicit descriptions of the target domain. ② The proliferation of ML-based techniques have more or less transformed the workflow of design automation, directly driving

rapid and agile hardware design. We expect GNNs make better use of naturally graphical data in EDA field; we expect deep RL is 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 to 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 computer architecture/systems roughly falls into two categories: ML-based fast modeling, which involves performance metrics or some other criteria of interest, and ML-based design methodology, which directly leverages ML as the design tool. 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 *Proceedings of the NIPS Machine Learning for Systems Workshop*.

[3] Nitish Agarwal, Tulsi Jain, and Mohamed Zahran. 2019. Performance prediction for multi-threaded applications. In *Proceedings of the International Workshop on AI-assisted Design for Architecture (AIDArc), Held in Conjunction with ISCA*.

[4] Byung Hoon Ahn, Prannoy Pilligundla, and Hadi Esmaeilzadeh. 2019. Reinforcement learning and adaptive sampling for optimized DNN compilation. *Arxiv Preprint arXiv:1905.12799* (2019).

[5] 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*. ACM, 9.

[6] Mohamed Baker Alawieh, Wuxi Li, Yibo Lin, Love Singhal, Mahesh A. Iyer, and David Z. Pan. 2020. High-definition routing congestion prediction for large-scale FPGAs. In *Proceedings of the 25th Asia and South Pacific Design Automation Conference (ASP-DAC)*. IEEE, 26–31.

[7] Miltiadis Allamanis, Earl T. Barr, Premkumar Devanbu, and Charles Sutton. 2018. A survey of machine learning for big code and naturalness. *ACM Comput. Surv.* 51, 4 (2018), 1–37.

[8] Naomi S. Altman. 1992. An introduction to kernel and nearest-neighbor nonparametric regression. *Amer. Statist.* 46, 3 (1992), 175–185.

[9] 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.

[10] 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).

[11] Yuxin Bai, Victor W. Lee, and Engin Ipek. 2017. Voltage regulator efficiency aware power management. *ACM SIGOPS Oper. Syst. Rev.* 51, 2 (2017), 825–838.

[12] 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 *Proceedings of the 43rd International Conference on Parallel Processing*. IEEE, 371–380.

[13] Bharathan Balaji, Christopher Kakovitch, and Balakrishnan Narayanaswamy. 2020. Fireplace: Placing firecracker virtual machines with hindsight imitation. In *Proceedings of the ML for Systems Workshop at NeurIPS 2020*.

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

[15] Luiz André Barroso, Urs Hölzle, and Parthasarathy Ranganathan. 2018. The datacenter as a computer: Designing warehouse-scale machines. *Synth. Lect. Comput. Archit.* 13, 3 (2018), i–189.

[16] Nathan Beckmann and Daniel Sanchez. 2017. Maximizing cache performance under uncertainty. In *Proceedings of the IEEE International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 109–120.
[17] 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.
[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 Comput. Archit. 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 *Proceedings of the 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 *Proceedings of the IEEE/ACM International Conference on Computer-aided Design (ICCAD)*. IEEE, 664–669.
[21] OpenAI Blog. 2018. *Al and Compute*. Retrieved from: https://openai.com/blog/ai-and-compute/.
[22] Shekhar Borkar. 2013. Exascale computing—A fact or affliction. *Keynote Present. IPDPS* 10 (2013).
[23] Justin A. Boyan and Michael L. Littman. 1994. Packet routing in dynamically changing networks: A reinforcement learning approach. In *Proceedings of the International Conference on Advances in Neural Information Processing Systems*. 671–678.
[24] Peter Braun and Heiner Litz. 2019. Understanding memory access patterns for prefetching. In *Proceedings of the International Workshop on AI-assisted Design for Architecture (AIDArc), Held in Conjunction with ISCA*.
[25] Jonathon Byrd and Zachary Lipton. 2019. What is the effect of importance weighting in deep learning? In *Proceedings of the International Conference on Machine Learning*. PMLR, 872–881.
[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 Trans. Program. Lang. Syst.* 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 Trans. Cloud Comput.* 3, 4 (2014), 449–458.
[28] Wei-Ting J. Chan, Yang Du, Andrew B. Kahng, Siddhartha Nath, and Kambiz Samadi. 2016. BEOL stack-aware routability prediction from placement using data mining techniques. In *Proceedings of the IEEE 34th International Conference on Computer Design (ICCD)*. IEEE, 41–48.
[29] Jingsong Chen, Jian Kuang, Guowei Zhao, Dennis J.-H. Huang, and Evangeline F. Y. Young. 2020. PROS: A plug-in for routability optimization applied in the state-of-the-art commercial EDA tool using deep learning. In *Proceedings of the IEEE/ACM International Conference on Computer Aided Design (ICCAD)*. IEEE, 1–8.
[30] 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 Conference of the ACM Special Interest Group on Data Communication*. 191–205.
[31] Xinyun Chen, Chang Liu, and Dawn Song. 2018. Tree-to-tree neural networks for program translation. *Adv. Neural Inf. Process. Syst.* 31 (2018), 2547–2557.
[32] Zhuo Chen and Diana Marculescu. 2015. Distributed reinforcement learning for power limited many-core system performance optimization. In *Proceedings of the 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 Trans. Comput.-aided Des. Integ. Circ. Syst.* 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 Trans. Comput.-aided Des. Integ. Circ. Syst.* 38, 9 (2018), 1717–1730.
[35] 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.
[36] Ryan Cochran, Can Hankendi, Ayse K. Coskun, and Sherief Reda. 2011. Pack & Cap: Adaptive DVFS and thread packing under power caps. In *Proceedings of the 44th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 175–185.
[37] 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.
[38] 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.

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

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

[41] 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.

[42] Dong Dai, Forrest Sheng Bao, Jiang Zhou, and Yong Chen. 2016. Block2Vec: A deep learning strategy on mining block correlations in storage systems. In *Proceedings of the 45th International Conference on Parallel Processing Workshops (ICPPW)*. IEEE, 230–239.

[43] Steve Dai, Yuan Zhou, Hang Zhang, Ecenur Ustun, Evangeline F. Y. Young, and Zhiru Zhang. 2018. Fast and accurate estimation of quality of results in high-level synthesis with machine learning. In *Proceedings of the IEEE 26th Annual International Symposium on Field-programmable Custom Computing Machines (FCCM)*. IEEE, 129–132.

[44] 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.

[45] 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.

[46] 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 *Proceedings of the 53rd ACM/EDAC/IEEE Design Automation Conference (DAC)*. IEEE, 1–6.

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

[48] 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.

[49] 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.

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

[51] 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.

[52] Xiangyu Dong, Norman P. Jouppi, and Yuan Xie. 2013. A circuit-architecture co-optimization framework for exploring nonvolatile memory hierarchies. *ACM Trans. Archit. Code Optim.* 10, 4 (2013), 23.

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

[54] 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 *Proceedings of the IEEE/ACM 6th International Symposium on Networks-on-chip*. IEEE, 19–26.

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

[56] 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.

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

[58] 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*.

[59] 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 3rd International Workshop on Network on Chip Architectures*. ACM, 11–16.

[60] 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).
[61] Chelsea Finn, Pieter Abbeel, and Sergey Levine. 2017. Model-agnostic meta-learning for fast adaptation of deep networks. In *Proceedings of the International Conference on Machine Learning*. PMLR, 1126–1135.
[62] Jerome H. Friedman. 1991. Multivariate adaptive regression splines. *Ann. Statist.* 19, 1 (1991), 1–141.
[63] 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 *Proceedings of the International Conference on Advances in Neural Information Processing Systems*. 3703–3714.
[64] Archana Ganapathi, Kaushik Datta, Armando Fox, and David Patterson. 2009. A case for machine learning to optimize multicore performance. In *Proceedings of the 1st USENIX Conference on Hot Topics in Parallelism*. USENIX Association Berkeley, CA, 1–1.
[65] Jim Gao. 2014. Machine learning applications for data center optimization. https://research.google/pubs/pub42542/
[66] Yuanxiang Gao, Li Chen, and Baochun Li. 2018. Spotlight: Optimizing device placement for training deep neural networks. In *Proceedings of the International Conference on Machine Learning*. 1662–1670.
[67] 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.
[68] Andrew Gelman, John B. Carlin, Hal S. Stern, David B. Dunson, Aki Vehtari, and Donald B. Rubin. 2013. *Bayesian Data Analysis*. CRC Press.
[69] Faustino J. Gomez, Doug Burger, and Risto Miikkulainen. 2001. A neuro-evolution method for dynamic resource allocation on a chip multiprocessor. In *Proceedings of the International Joint Conference on Neural Networks*. IEEE, 2355–2360.
[70] Zhenhuan Gong, Xiaohui Gu, and John Wilkes. 2010. Press: Predictive elastic resource scaling for cloud systems. In *Proceedings of the International Conference on Network and Service Management*. IEEE, 9–16.
[71] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. 2016. *Deep Learning*. The MIT Press.
[72] David Ha, Andrew Dai, and Quoc V. Le. 2016. Hypernetworks. *Arxiv Preprint arXiv:1609.09106* (2016).
[73] Winston Haaswijk, Edo Collins, Benoit Seguin, Mathias Soeken, Frédéric Kaplan, Sabine Süsstrunk, and Giovanni De Micheli. 2018. Deep learning for logic optimization algorithms. In *Proceedings of the IEEE International Symposium on Circuits and Systems (ISCAS)*. IEEE, 1–4.
[74] Kourosh Hakhamaneshi, Nick Werblun, Pieter Abbeel, and Vladimir Stojanović. 2019. BagNet: Berkeley analog generator with layout optimizer boosted with deep neural networks. In *Proceedings of the IEEE/ACM International Conference on Computer-aided Design (ICCAD)*. IEEE, 1–8.
[75] Greg Hamerly, Charles Elkan, et al. 2001. Bayesian approaches to failure prediction for disk drives. In *Proceedings of the International Conference on Machine Learning*. 202–209.
[76] Song Han, Huizi Mao, and William J. Dally. 2016. Deep compression: Compressing deep neural networks with pruning, trained quantization, and Huffman coding. In *Proceedings of the International Conference on Learning Representations*.
[77] Milad Hashemi, Kevin Swersky, Jamie Smith, Grant Ayers, Heiner Litz, Jichuan Chang, Christos Kozyrakis, and Parthasarathy Ranganathan. 2018. Learning memory access patterns. In *Proceedings of the International Conference on Machine Learning*. 1924–1933.
[78] 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).
[79] Henry Hoffmann. 2015. JouleGuard: Energy guarantees for approximate applications. In *Proceedings of the 25th Symposium on Operating Systems Principles*. 198–214.
[80] David W. Hosmer Jr, Stanley Lemeshow, and Rodney X. Sturdivant. 2013. *Applied Logistic Regression*. Vol. 398. John Wiley & Sons.
[81] Abdelrahman Hosny, Soheil Hashemi, Mohamed Shalan, and Sherief Reda. 2020. DRiLLS: Deep reinforcement learning for logic synthesis. In *Proceedings of the 25th Asia and South Pacific Design Automation Conference (ASP-DAC)*. IEEE, 581–586.
[82] 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.
[83] Engin Ïpek, Sally A. McKee, Rich Caruana, Bronis R. de Supinski, and Martin Schulz. 2006. *Efficiently Exploring Architectural Design Spaces Via Predictive Modeling*. Vol. 41. ACM.
[84] E. Ipek, O. Mutlu, J. F. Martinez, and R. Caruana. 2008. Self-optimizing memory controllers: A reinforcement learning approach. In *Proceedings of the International Symposium on Computer Architecture (ISCA)*. ACM.
[85] Sadeka Islam, Jacky Keung, Kevin Lee, and Anna Liu. 2012. Empirical prediction models for adaptive resource provisioning in the cloud. *Fut. Gen. Comput. Syst.* 28, 1 (2012), 155–162.

[86] 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.

[87] 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)*.

[88] Anil K. Jain. 2010. Data clustering: 50 years beyond k-means. *Pattern Recog. Lett.* 31, 8 (2010), 651–666.

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

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

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

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

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

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

[95] Daniel A. Jiménez and Calvin Lin. 2001. Dynamic branch prediction with perceptrons. In *Proceedings of the 7th International Symposium on High-performance Computer Architecture*. IEEE, 197–206.

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

[97] 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.

[98] Albin Jones, Amir Yazdanbakhsh, Berkin Akin, Christof Angermueller, James Pierce Laudon, Kevin Swersky, Milad Hashemi, Ravi Narayanaswami, Sat Chatterjee, and Yanqi Zhou. 2020. Apollo: Transferable architecture exploration. In *Proceedings of the ML for Systems Workshop at NeurIPS 2020*.

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

[100] P. J. Joseph, Kapil Vaswani, and Matthew J. Thazhuthaveetil. 2006. Construction and use of linear regression models for processor performance analysis. In *Proceedings of the 12th International Symposium on High-performance Computer Architecture*. IEEE, 99–108.

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

[102] 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 9th IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis*. IEEE Press, 8.

[103] Da-Cheng Juan and Diana Marculescu. 2012. Power-aware performance increase via core/uncore reinforcement control for chip-multiprocessors. In *Proceedings of the ACM/IEEE International Symposium on Low Power Electronics and Design*. 97–102.

[104] Wonkyung Kang, Dongkun Shin, and Sungjoo Yoo. 2017. Reinforcement learning-assisted garbage collection to mitigate long-tail latency in SSD. *ACM Trans. Embed. Comput. Syst.* 16, 5s (2017), 1–20.

[105] 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 *Proceedings of the 55th ACM/ESDA/IEEE Design Automation Conference (DAC)*. IEEE, 1–6.

[106] Sheng-Chun Kao, Geonhwa Jeong, and Tushar Krishna. 2020. ConfuciuX: Autonomous hardware resource assignment for DNN accelerators using reinforcement learning. In *Proceedings of the 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 622–636.

[107] Shauharda Khadka, Estelle Aflalo, Mattias Marder, Avrech Ben-David, Santiago Miret, Hanlin Tang, Shie Mannor, Tamir Hazan, and Somdeb Majumdar. 2020. Optimizing memory placement using evolutionary graph reinforcement learning. *Arxiv Preprint arXiv:2007.07298* (2020).

[108] 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.

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

[110] Salman Khan, Polychronis Xekalakis, John Cavazos, and Marcelo Cintra. 2007. Using predictive modeling for cross-program design space exploration in multicore systems. In *Proceedings of the 16th International Conference on Parallel Architecture and Compilation Techniques (PACT'07)*. IEEE, 327–338.

[111] Yonghae Kim and Hyesoon Kim. 2019. A case study: Exploiting neural machine translation to translate CUDA to OpenCL. *Arxiv Preprint arXiv:1905.07653* (2019).

[112] David Koeplinger, Raghu Prabhakar, Yaqi Zhang, Christina Delimitrou, Christos Kozyrakis, and Kunle Olukotun. 2016. Automatic generation of efficient accelerators for reconfigurable hardware. In *Proceedings of the ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 115–127.

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

[114] 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.

[115] 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.

[116] 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*.

[117] Ravi Kumar, Manish Purohit, Zoya Svitkina, and Erik Vee. 2020. Interleaved caching with access graphs. In *Proceedings of the 14th Annual ACM-SIAM Symposium on Discrete Algorithms*. SIAM, 1846–1858.

[118] 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.

[119] Jihye Kwon and Luca P. Carloni. 2020. Transfer learning for design-space exploration with high-level synthesis. In *Proceedings of the ACM/IEEE Workshop on Machine Learning for CAD*. 163–168.

[120] Marie-Anne Lachaux, Baptiste Roziere, Lowik Chanussot, and Guillaume Lample. 2020. Unsupervised translation of programming languages. *Arxiv Preprint arXiv:2006.03511* (2020).

[121] 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.

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

[123] 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.

[124] 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.

[125] Bowen Li and Paul D. Franzon. 2016. Machine learning in physical design. In *Proceedings of the IEEE 25th Conference on Electrical Performance of Electronic Packaging and Systems (EPEPS)*. IEEE, 147–150.

[126] 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 *Proceedings of the 44th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*. IEEE, 383–394.

[127] Jing Li, Rebecca J. Stones, Gang Wang, Xiaoguang Liu, Zhongwei Li, and Ming Xu. 2017. Hard drive failure prediction using decision trees. *Reliab. Eng. Syst. Safety* 164 (2017), 55–65.

[128] Yaguang Li, Yishuang Lin, Meghna Madhusudan, Arvind Sharma, Wenbin Xu, Sachin Sapatnekar, Ramesh Harjani, and Jiang Hu. 2020. Exploring a machine learning approach to performance driven analog IC placement. In *Proceedings of the IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*. IEEE, 24–29.

[129] 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 *Proceedings of the IEEE/ACM International Conference on Computer Aided Design (ICCAD)*. IEEE, 1–9.

[130] Yunfan Li, D. Penney, Abhishek Ramamurthy, and Lizhong Chen. 2019. Characterizing on-chip traffic patterns in general-purpose GPUs: A deep learning approach. In *Proceedings of the International Conference on Computer Design (ICCD)*.

[131] Rongjian Liang, Hua Xiang, Diwesh Pandey, Lakshmi Reddy, Shyam Ramji, Gi-Joon Nam, and Jiang Hu. 2020. DRC hotspot prediction at sub-10nm process nodes using customized convolutional network. In *Proceedings of the International Symposium on Physical Design*. 135–142.

[132] 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).

[133] 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 Comput. Archit. Lett.* 18, 1 (2019), 51–54.

[134] 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).

[135] Yibo Lin, Shounak Dhar, Wuxi Li, Haoxing Ren, Brucek Khailany, and David Z. Pan. 2019. DREAMPlace: Deep learning toolkit-enabled GPU acceleration for modern VLSI placement. In *Proceedings of the 56th ACM/IEEE Design Automation Conference (DAC)*. IEEE, 1–6.

[136] Zhe Lin, Jieru Zhao, Sharad Sinha, and Wei Zhang. 2020. HL-Pow: A learning-based power modeling framework for high-level synthesis. In *Proceedings of the 25th Asia and South Pacific Design Automation Conference (ASP-DAC)*. IEEE, 574–580.

[137] Hung-Yi Liu and Luca P. Carloni. 2013. On learning-based methods for design-space exploration with high-level synthesis. In *Proceedings of the 50th Annual Design Automation Conference*. 1–7.

[138] Mingjie Liu, Keren Zhu, Jiaqi Gu, Linxiao Shen, Xiyuan Tang, Nan Sun, and David Z. Pan. 2020. Towards decrypting the art of analog layout: Placement quality prediction via transfer learning. In *Proceedings of the Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 496–501.

[139] Mingjie Liu, Keren Zhu, Xiyuan Tang, Biying Xu, Wei Shi, Nan Sun, and David Z. Pan. 2020. Closing the design loop: Bayesian optimization assisted hierarchical analog layout synthesis. In *Proceedings of the 57th ACM/IEEE Design Automation Conference (DAC)*. IEEE, 1–6.

[140] 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.

[141] Wei-Yin Loh. 2011. Classification and regression trees. *Wiley Interdisc. Rev.: Data Mining Knowl. Discov.* 1, 1 (2011), 14–23.

[142] 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.

[143] Yi-Chen Lu, Jeehyun Lee, Anthony Agnesina, Kambiz Samadi, and Sung Kyu Lim. 2019. GAN-CTS: A generative adversarial framework for clock tree prediction and optimization. In *Proceedings of the IEEE/ACM International Conference on Computer-aided Design (ICCAD)*. IEEE, 1–8.

[144] 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 *Proceedings of the 41st International Conference on Parallel Processing*. IEEE, 48–57.

[145] Dani Maarouf, Abeer Alhyari, Ziad Abuowaimer, Timothy Martin, Andrew Gunter, Gary Grewal, Shawki Areibi, and Anthony Vannelli. 2018. Machine-learning based congestion estimation for modern FPGAs. In *Proceedings of the 28th International Conference on Field Programmable Logic and Applications (FPL)*. IEEE, 427–4277.

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

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

[148] Hosein Mohammadi Makrani, Farnoud Farahmand, Hossein Sayadi, Sara Bondi, Sai Manoj Pudukotai Dinakarrao, Houman Homayoun, and Setareh Rafatirad. 2019. Pyramid: Machine learning framework to estimate the optimal timing and resource usage of a high-level synthesis design. In *Proceedings of the 29th International Conference on Field Programmable Logic and Applications (FPL)*. IEEE, 397–403.

[149] Hosein Mohammadi Makrani, Hossein Sayadi, Tinoosh Mohsenin, Setareh Rafatirad, Avesta Sasan, and Houman Homayoun. 2019. XPPE: Cross-platform performance estimation of hardware accelerators using machine learning. In *Proceedings of the 24th Asia and South Pacific Design Automation Conference*. 727–732.

[150] 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.

[151] Hongzi Mao, Malte Schwarzkopf, Shaileshh Bojja Venkatakrishnan, Zili Meng, and Mohammad Alizadeh. 2019. Learning scheduling algorithms for data processing clusters. In *Proceedings of the ACM Special Interest Group on Data Communication*. 270–288.

[152] Artemiy Margaritov, Dmitrii Ustiugov, Edouard Bugnion, and Boris Grot. 2018. Virtual address translation via learned page table indexes. In *Proceedings of the 32nd Conference on Neural Information Processing Systems (NeurIPS)*.

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

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

[155] Atefeh Mehrabi, Aninda Manocha, Benjamin C. Lee, and Daniel J. Sorin. 2020. Prospector: Synthesizing efficient accelerators via statistical learning. In *Proceedings of the Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 151–156.

[156] Charith Mendis and Saman Amarasinghe. 2018. goSLP: Globally optimized superword level parallelism framework. *Proc. ACM Program. Lang.* 2, OOPSLA (2018), 1–28.

[157] Charith Mendis, Alex Renda, Saman Amarasinghe, and Michael Carbin. 2019. Ithemal: Accurate, portable and fast basic block throughput estimation using deep neural networks. In *Proceedings of the International Conference on Machine Learning*. 4505–4515.

[158] Charith Mendis, Cambridge Yang, Yewen Pu, Saman Amarasinghe, and Michael Carbin. 2019. Compiler auto-vectorization with imitation learning. In *Proceedings of the International Conference on Advances in Neural Information Processing Systems*. 14598–14609.

[159] Pingfan Meng, Alric Althoff, Quentin Gautier, and Ryan Kastner. 2016. Adaptive threshold non-Pareto elimination: Re-thinking machine learning for system level design space exploration on FPGAs. In *Proceedings of the Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 918–923.

[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*. 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 23rd 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 *Proceedings of the 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 *Proceedings of the International Conference on Machine Learning*. PMLR, 1928–1937.

[167] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, Georg Ostrovski et al. 2015. Human-level control through deep reinforcement learning. *Nature* 518, 7540 (2015), 529–533.

[168] Gordon E. Moore et al. 1965. Cramming more components onto integrated circuits.

[169] Janani Mukundan and Jose F. Martinez. 2012. MORSE: Multi-objective reconfigurable self-optimizing memory scheduler. In *Proceedings of the IEEE International Symposium on High-performance Computer Architecture*. IEEE, 1–12.

[170] Joseph F. Murray, Gordon F. Hughes, and Kenneth Kreutz-Delgado. 2005. Machine learning methods for predicting failures in hard drives: A multiple-instance application. *J. Mach. Learn. Res.* 6, May (2005), 783–816.

[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 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 *Proceedings of the 29th International Symposium on Computer Architecture and High Performance Computing (SBAC-PAD)*. IEEE, 121–128.

[173] Walter Lau Neto, Max Austin, Scott Temple, Luca Amaru, Xifan Tang, and Pierre-Emmanuel Gaillardon. 2019. LSOracle: A logic synthesis framework driven by artificial intelligence. In *Proceedings of the IEEE/ACM International Conference on Computer-aided Design (ICCAD)*. IEEE, 1–6.

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

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

[176] Kenneth O'Neal, Mitch Liu, Hans Tang, Amin Kalantar, Kennen DeRenard, and Philip Brisk. 2018. HLSPredict: Cross platform performance prediction for FPGA high-level synthesis. In *Proceedings of the IEEE/ACM International Conference on Computer-aided Design (ICCAD)*. IEEE, 1–8.

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

[178] Gung-Yu Pan, Jing-Yang Jou, and Bo-Cheng Lai. 2014. Scalable power management using multilevel reinforcement learning for multiprocessors. *ACM Trans. Des. Automat. Electron. Syst.* 19, 4 (2014), 33.

[179] Po-Cheng Pan, Chien-Chia Huang, and Hung-Ming Chen. 2019. Late breaking results: An efficient learning-based approach for performance exploration on analog and RF circuit synthesis. In *Proceedings of the 56th ACM/IEEE Design Automation Conference (DAC)*. IEEE, 1–2.

[180] Ghasem Pasandi, Shahin Nazarian, and Massoud Pedram. 2019. Approximate logic synthesis: A reinforcement learning-based technology mapping approach. In *Proceedings of the 20th International Symposium on Quality Electronic Design (ISQED)*. IEEE, 26–32.

[181] 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 International Conference on Parallel Architectures and Compilation*. ACM, 31–44.

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

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

[184] Zhongdong Qi, Yici Cai, and Qiang Zhou. 2014. Accurate prediction of detailed routing congestion using supervised data learning. In *Proceedings of the IEEE 32nd International Conference on Computer Design (ICCD)*. IEEE, 97–103.

[185] 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 *Proceedings of the Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 354–357.

[186] Nishant Rao, Akshay Ramachandran, and Amish Shah. 2018. MLNoC: A machine learning based approach to NoC design. In *Proceedings of the 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 *Proceedings of the 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] Haoxing Ren, Matthew Fojtik, and Brucek Khailany. 2020. NVCell: Generate standard cell layout in advanced technology nodes with reinforcement learning. In *Proceedings of the ML for Systems Workshop at NeurIPS 2020*.

[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 *Proceedings of the 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 *Proceedings of the IEEE International Symposium on Circuits and Systems (ISCAS)*. IEEE, 1–5.

[193] Christian Ritz and Jens Carl Streibig. 2008. *Nonlinear Regression with R*. Springer Science & Business Media.

[194] João P. S. Rosa, Daniel J. D. Guerra, Nuno C. G. Horta, Ricardo M. F. Martins, and Nuno C. C. Lourenço. 2019. *Using Artificial Neural Networks for Analog Integrated Circuit Design Automation*. Springer Nature.

[195] Michael Rotman and Lior Wolf. 2020. Electric analog circuit design with hypernetworks and a differential simulator. In *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 4157–4161.

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

[197] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. 1985. *Learning Internal Representations by Error Propagation.* Technical Report. California University San Diego La Jolla Institute for Cognitive Science.
[198] Jaehun Ryu and Hyojin Sung. 2021. MetaRune: Meta-learning based cost model for fast and efficient auto-tuning frameworks. *Arxiv Preprint arXiv:2102.04199* (2021).
[199] Omer Sagi and Lior Rokach. 2018. Ensemble learning: A survey. *Wiley Interdisc. Rev.: Data Mining Knowl. Discov.* 8, 4 (2018), e1249.
[200] 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.
[201] Andreas G. Savva, Theocharis Theocharides, and Vassos Soteriou. 2012. Intelligent on/off dynamic link management for on-chip networks. *J. Electric. Comput. Eng.* 2012 (2012), 6.
[202] Bernhard Schölkopf, Alexander J. Smola, Francis Bach et al. 2002. *Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond.* The MIT Press.
[203] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. 2017. Proximal policy optimization algorithms. *Arxiv Preprint arXiv:1707.06347* (2017).
[204] George A. F. Seber and Alan J. Lee. 2012. *Linear Regression Analysis.* Vol. 329. John Wiley & Sons.
[205] Keertana Settaluri, Ameer Haj-Ali, Qijing Huang, Kourosh Hakhamaneshi, and Borivoje Nikolic. 2020. AutoCkt: Deep reinforcement learning of analog circuit designs. In *Proceedings of the Design, Automation & Test in Europe Conference & Exhibition (DATE).* IEEE, 490–495.
[206] Burr Settles. 2009. Active learning literature survey. (2009).
[207] 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.
[208] 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.]).
[209] Zhan Shi, Kevin Swersky, Daniel Tarlow, Parthasarathy Ranganathan, and Milad Hashemi. 2019. Learning execution through neural code fusion. *Arxiv Preprint arXiv:1906.07181* (2019).
[210] Brett Shook, Prateek Bhansali, Chandramouli Kashyap, Chirayu Amin, and Siddhartha Joshi. 2020. MLParest: Machine learning based parasitic estimation for custom circuit design. In *Proceedings of the 57th ACM/IEEE Design Automation Conference (DAC).* IEEE, 1–6.
[211] Zhenyu Song, Daniel S. Berger, Kai Li, Anees Shaikh, Wyatt Lloyd, Soudeh Ghorbani, Changhoon Kim, Aditya Akella, Arvind Krishnamurthy, Emmett Witchel, et al. 2020. Learning relaxed Belady for content distribution network caching. In *Proceedings of the 17th USENIX Symposium on Networked Systems Design and Implementation (NSDI'20).* 529–544.
[212] 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.
[213] 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.
[214] Richard S. Sutton and Andrew G. Barto. 2018. *Reinforcement Learning: An Introduction.* The MIT Press.
[215] 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.
[216] Aysa Fakheri Tabrizi, Logan Rakai, Nima Karimpour Darav, Ismail Bustany, Laleh Behjat, Shuchang Xu, and Andrew Kennings. 2018. A machine learning framework to identify detailed routing short violations from a placed netlist. In *Proceedings of the 55th ACM/ESDA/IEEE Design Automation Conference (DAC).* IEEE, 1–6.
[217] 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.
[218] 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).
[219] Elvira Teran, Zhe Wang, and Daniel A. Jiménez. 2016. Perceptron learning for reuse prediction. In *Proceedings of the 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO).* IEEE, 1–12.
[220] Gerald Tesauro. 2007. Reinforcement learning in autonomic computing: A manifesto and case studies. *IEEE Internet Comput.* 11, 1 (2007), 22–30.
[221] Gerald Tesauro et al. 2005. Online resource allocation using decompositional reinforcement learning. In *Proceedings of the Association for the Advancement of Artificial Intelligence Conference.* 886–891.

[222] Alexey Tsymbal. 2004. The problem of concept drift: Definitions and related work. *Comput. Sci. Depart., Trinity Coll. Dublin* 106, 2 (2004), 58.

[223] Ecenur Ustun, Chenhui Deng, Debjit Pal, Zhijing Li, and Zhiru Zhang. 2020. Accurate operation delay prediction for FPGA HLS using graph neural networks. In *Proceedings of the 39th International Conference on Computer-aided Design*. 1–9.

[224] 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 *Proceedings of the IEEE International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 480–491.

[225] Joaquin Vanschoren. 2018. Meta-learning: A survey. *Arxiv Preprint arXiv:1810.03548* (2018).

[226] David Vengerov. 2009. A reinforcement learning framework for utility-based scheduling in resource-constrained systems. *Fut. Gen. Comput. Syst.* 25, 7 (2009), 728–736.

[227] 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.

[228] M. Mitchell Waldrop. 2016. The chips are down for Moore's law. *Nat. News* 530, 7589 (2016), 144.

[229] 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*. ACM, 46.

[230] Haoyuan Wang and Zhiwei Luo. 2017. Data cache prefetching with perceptron learning. *Arxiv Preprint arXiv:1712.00905* (2017).

[231] 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 *Proceedings of the 57th ACM/IEEE Design Automation Conference (DAC)*. IEEE, 1–6.

[232] 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.

[233] Ke Wang, Ahmed Louri, Avinash Karanth, and Razvan Bunescu. 2019. High-performance, energy-efficient, fault-tolerant network-on-chip design using reinforcement learning. In *Proceedings of the Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 1166–1171.

[234] 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.

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

[236] Shimon Whiteson and Peter Stone. 2004. Adaptive job routing and scheduling. *Eng. Applic. Artif. Intell.* 17, 7 (2004), 855–869.

[237] Svante Wold, Kim Esbensen, and Paul Geladi. 1987. Principal component analysis. *Chemomet. Intell. Lab. Syst.* 2, 1-3 (1987), 37–52.

[238] 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 *Proceedings of the IEEE 20th International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 308–319.

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

[240] 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.

[241] 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 Trans. Des. Automat. Electron. Syst.* 26, 2 (2020), 1–27.

[242] Nan Wu and Pengcheng Li. 2020. Phoebe: Reuse-aware online caching with reinforcement learning for emerging storage models. *Arxiv Preprint arXiv:2011.07160* (2020).

[243] Nan Wu, Yuan Xie, and Cong Hao. 2021. Ironman: GNN-assisted design space exploration in high-level synthesis via reinforcement learning. In *Proceedings of the Great Lakes Symposium on VLSI*. 39–44.

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

[245] Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and S. Yu Philip. 2020. A comprehensive survey on graph neural networks. *IEEE Trans. Neural Netw. Learn. Syst.* 32, 1 (2020), 4–24.

[246] 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.

[247] Zhiyao Xie, Yu-Hung Huang, Guan-Qi Fang, Haoxing Ren, Shao-Yun Fang, Yiran Chen, and Jiang Hu. 2018. RouteNet: Routability prediction for mixed-size designs using convolutional neural network. In *Proceedings of the IEEE/ACM International Conference on Computer-aided Design (ICCAD)*. IEEE, 1–8.

[248] 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.

[249] 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 *Proceedings of the USENIX Annual Technical Conference (USENIX ATC'18)*. 481–494.

[250] Que Yanghua, Nachiket Kapre, Harnhua Ng, and Kirvy Teo. 2016. Improving classification accuracy of a machine learning approach for FPGA timing closure. In *Proceedings of the IEEE 24th Annual International Symposium on Field-programmable Custom Computing Machines (FCCM)*. IEEE, 80–83.

[251] 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.

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

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

[254] Jieming Yin, Yasuko Eckert, Shuai Che, Mark Oskin, and Gabriel H. Loh. 2018. Toward more efficient NoC arbitration: A deep reinforcement learning approach. (2018).

[255] 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 *Proceedings of the IEEE International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 637–648.

[256] Cunxi Yu, Houping Xiao, and Giovanni De Micheli. 2018. Developing synthesis flows without human knowledge. In *Proceedings of the 55th Annual Design Automation Conference*. 1–6.

[257] Cunxi Yu and Zhiru Zhang. 2019. Painting on placement: Forecasting routing congestion using conditional generative adversarial nets. In *Proceedings of the 56th Annual Design Automation Conference*. 1–6.

[258] Cunxi Yu and Wang Zhou. 2020. Decision making in synthesis cross technologies using LSTMs and transfer learning. In *Proceedings of the ACM/IEEE Workshop on Machine Learning for CAD*. 55–60.

[259] Georgios Zacharopoulos, Andrea Barbon, Giovanni Ansaloni, and Laura Pozzi. 2018. Machine learning approach for loop unrolling factor prediction in high level synthesis. In *Proceedings of the International Conference on High Performance Computing & Simulation (HPCS)*. IEEE, 91–97.

[260] 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 *Proceedings of the 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 1–12.

[261] 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.

[262] Guo Zhang, Hao He, and Dina Katabi. 2019. Circuit-GNN: Graph neural networks for distributed circuit design. In *Proceedings of the International Conference on Machine Learning*. PMLR, 7364–7373.

[263] 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.

[264] 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).

[265] Jieru Zhao, Tingyuan Liang, Sharad Sinha, and Wei Zhang. 2019. Machine learning based routing congestion prediction in FPGA high-level synthesis. In *Proceedings of the Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 1130–1135.

[266] 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*. ACM, 47.

[267] Pengfei Zheng and Benjamin C. Lee. 2018. Hound: Causal learning for datacenter-scale straggler diagnosis. *Proc. ACM Measur. Anal. Comput. Syst.* 2, 1 (2018), 1–36.

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

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

[270] 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).

[271] Bingpeng Zhu, Gang Wang, Xiaoguang Liu, Dianming Hu, Sheng Lin, and Jingwei Ma. 2013. Proactive drive failure prediction for large scale storage systems. In *Proceedings of the IEEE 29th Symposium on Mass Storage Systems and Technologies (MSST)*. IEEE, 1–5.

[272] Keren Zhu, Mingjie Liu, Yibo Lin, Biying Xu, Shaolan Li, Xiyuan Tang, Nan Sun, and David Z. Pan. 2019. Geniusroute: A new analog routing paradigm using generative neural network guidance. In *Proceedings of the IEEE/ACM International Conference on Computer-aided Design (ICCAD)*. IEEE, 1–8.

[273] Xiaojin Jerry Zhu. 2005. *Semi-supervised Learning Literature Survey*. Technical Report. University of Wisconsin-Madison Department of Computer Sciences.

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

Received March 2021; revised August 2021; accepted October 2021