MLCAD: A Survey of Research in Machine Learning for CAD *Keynote Paper*

Martin Rapp, *Graduate Student Member, IEEE*, Hussam Amrouch, *Member, IEEE*, Yibo Lin, *Member, IEEE*, Bei Yu, *Member, IEEE*, David Z. Pan, *Fellow, IEEE*, Marilyn Wolf, *Fellow, IEEE*, and Jörg Henkel, *Fellow, IEEE*

***Abstract*—Due to the increasing size of integrated circuits (ICs), their design and optimization phases (i.e., computer-aided design, CAD) grow increasingly complex. At design time, a large design space needs to be explored to find an implementation that fulfills all specifications and then optimizes metrics like energy, area, delay, reliability, etc. At run time, a large configuration space needs to be searched to find the best set of parameters (e.g., voltage/frequency) to further optimize the system. Both spaces are infeasible for exhaustive search typically leading to heuristic optimization algorithms that find some tradeoff between design quality and computational overhead. Machine learning (ML) can build powerful models that have successfully been employed in related domains. In this survey, we categorize how ML may be used and is used for design-time and run-time optimization and exploration strategies of ICs. A metastudy of published techniques unveils areas in computer-aided design (CAD) that are well explored and underexplored with ML, as well as trends in the employed ML algorithms. We present a comprehensive categorization and summary of the state of the art on ML for CAD. Finally, we summarize the remaining challenges and promising open research directions.**

***Index Terms*—Computer-aided design (CAD), deep learning, electronic design automation, machine learning (ML).**

Manuscript received 3 February 2021; revised 26 July 2021; accepted 14 October 2021. Date of publication 2 November 2021; date of current version 20 September 2022. This work was supported in part by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) through TRR 89 Invasive Computing under Project 146371743. The work of Bei Yu was supported by the Research Grants Council of Hong Kong SAR under Grant CUHK14209420. The work of David Z. Pan was supported by the U.S. National Science Foundation under Grant 1704758, Grant 1718570, and Grant 2112665, and in part by the DARPA IDEA Program. The work of Marilyn Wolf was supported by the U.S. National Science Foundation under Grant 2002853. This article was recommended by Associate Editor F. Liu. *(Corresponding author: Martin Rapp.)*

Martin Rapp and Jörg Henkel are with the Department of Computer Science, Karlsruhe Institute of Technology, 76131 Karlsruhe, Germany (e-mail: martin.rapp@kit.edu; henkel@kit.edu).

Hussam Amrouch is with the Department of Computer Science, University of Stuttgart, 70174 Stuttgart, Germany (e-mail: amrouch@iti.uni-stuttgart.de).

Yibo Lin is with the Department of Computer Science, Peking University, Beijing 100871, China (e-mail: yibolin@pku.edu.cn).

Bei Yu is with the Department of Computer Science and Engineering, The Chinese University of Hong Kong, Hong Kong (e-mail: byu@cse.cuhk.edu.hk).

David Z. Pan is with the Department of Electrical and Computer Engineering, The University of Texas at Austin, Austin, Austin, TX 78705 USA (e-mail: dpan@ece.utexas.edu).

Marilyn Wolf is with the Department of Computer Science & Engineering, University of Nebraska–Lincoln, Lincoln, NE 68588 USA (e-mail: mwolf@unl.edu).

Digital Object Identifier 10.1109/TCAD.2021.3124762

I. Introduction

THE COMPLEXITY of integrated circuits (ICs) continues to increase, mainly enabled by technology advances [1]. Therefore, the design and optimization of such systems for metrics, such as energy, area, delay, reliability, etc., both at design time and run time become more and more difficult. Still following Moore's law, the number of transistors per design increases exponentially and doubles every two years. Consequently, the corresponding design space, which needs to be searched for an implementation that fulfills all specifications and then optimizes the above-mentioned metrics, explodes. Analogously, the number of possible management actions at run time increases. Applications execute on an increasing number of processor cores that each needs to be operated at a certain voltage/frequency (v/f)-level – leading to more degrees of freedom that need to be exploited by run-time management to optimize the IC [2]. Both the design-time and run-time spaces are too large for exhaustive search. This has led to the development of a plethora of heuristic algorithms. However, such algorithms tend to suffer from low adaptability, and tend to either oversimplify the problem, leading to a low decision quality, or result in excessive computational complexity. The existing algorithms are not able to keep up with the high pace of technology advances, which manifests itself as the *design productivity gap*.

Machine learning (ML) techniques have been employed in many domains with great success because of their ability to build powerful models from data [3], [4]. Consequently, ML has also been applied in computer engineering, such as in computer-aided design (CAD) [5], where ML promises to fill the gaps left by heuristic algorithms and open new possibilities. Employing ML techniques allows designers to raise the abstraction level by focusing on the objective itself and leave the technical details on how to reach the objective to the ML model. For example, when optimizing lithographic masks with subresolution assist features (SRAFs) with ML, the designer specifies the goal (desired geometric pattern) but does not need to specify rules for where to place SRAFs. Another example is run-time management with reinforcement learning (RL), where the designer expresses the goals (e.g., high performance) with the reward function but does not to specify rules for when

which management action is to be executed.[1] This allows designers to handle more complex designs—mitigating the design productivity gap.

This survey provides a comprehensive summary of how ML techniques may be used and are used for CAD at various abstraction levels. We discuss both offline, design-time and online, run-time aspects of CAD and online (run-time) techniques because both are necessary to achieve design goals such as low-energy operation. We demonstrate the similarities in problems that are solved at design time and run time, as well as similarities in the employed ML models. Furthermore, many open challenges apply to both domains.

It is important to note that we only focus on techniques that use ML to design and optimize the IC itself. We explicitly do not include techniques that optimize ML training or inference for a user-application (e.g., accelerators), or ML techniques that solve a user-task (e.g., stroke detection).

Structure of This Survey: We first present a meta-study that analyzes all publications in five key conferences and journals in the area of CAD. This metastudy shows how many works employ ML for CAD and further breaks down these works to unveil trends in ML for CAD. We then present general patterns in how ML models can be employed in CAD. The identified patterns apply to both design-time and run-time techniques and demonstrate that similar ML models are applicable for both domains. The main part of this study gives an overview of all areas of design-time and run-time CAD to discuss the recent progress. Finally, we discuss open challenges and promising directions.

II. Trends in Machine Learning for CAD

This section presents a metastudy of how ML has been employed in CAD in the recent years. We analyzed all publications in the following venues: IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD), International Conference on CAD (ICCAD), Design Automation Conference (DAC), Asia and South Pacific Design Automation Conference (ASP-DAC), and the CAD conferences included in the Embedded Systems Week (International Conference on Compilers, Architecture, and Synthesis for Embedded Systems, CASES, and International Conference on Hardware/Software Codesign and System Synthesis, CODES+ISSS). We consider the criteria explained in the introduction, i.e., ML is used *for CAD*. This criterion excludes a large number of works that use ML as an application (accelerators, approximate computing, etc.). Only regular papers are considered while invited papers are excluded. We study the years from 2016 to 2020. This metastudy answers the following main questions.

1) Which are the areas in CAD that are well explored/not yet explored with ML?
2) Which ML algorithms have been used?
3) Which are the observable trends?

We divide CAD for ICs into the following six major design steps.

1) *System-level design space exploration (DSE) and high-level synthesis (HLS)* transform a high-level specification of the IC to register-transfer level (RTL) description. Thereby, HLS focuses on functional properties and system-level DSE optimizes nonfunctional properties. These steps determine the system architecture like decision which functions to implement in hardware or software, processor configurations, allocation of function units, scheduling, and binding of operations.
2) *Logic Synthesis* transforms the RTL description of a circuit to a gate-level representation in the target technology.
3) *Physical Design* includes placement of the logic gates on the die, routing of the connecting nets, design of the clock trees, and building a power/ground network. The output of physical design is a geometrical representation of the circuit.
4) *Manufacturing* the IC involves creating lithographic masks and the fabrication steps. Only a certain fraction of fabricated devices are functional due to process variations. This fraction is denoted as the yield.
5) *Verification and Test* ensures that fabricated devices adhere to the specifications. This involves testing of fabricated devices, but also verification techniques at earlier design steps to verify the correctness of the intermediate representations w.r.t. the specifications.
6) *Run-Time Management* dynamically adjusts parameters of the design like voltage or frequency according to the operating conditions.

[1]These examples are described in more details in Sections IV and V.

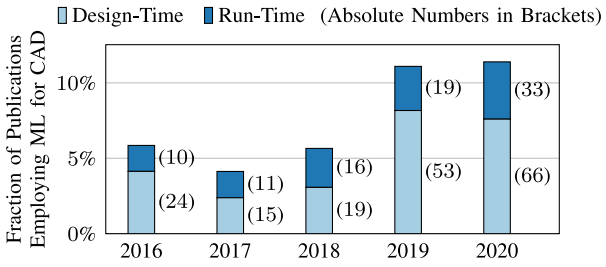

Fig. 1. Fraction of publications that employ ML for CAD among all regular papers published in *IEEE* TCAD, ICCAD, DAC, ASP-DAC, and ESWeek is growing strongly, increasing by almost 2× from 2016 to 2020.

Fig. 1 shows the fraction of publications that employ ML for CAD among all regular papers in the studied venues. It is apparent that ML techniques are gaining attraction. The fraction of publications increased by about 2× from 2016 to 2020 and reached 11% of all regular papers. Accordingly, the absolute numbers of publications increased. Around two thirds of these publications target the design-time steps 1–5, one-third target run-time management. The following analyses further break down these publications to answer our main questions.

Fig. 2 shows how many publications target individual *design-time steps*. Some works target several steps. In these cases, we assign them to the step by which the work is most extensively evaluated. In 2020, about 65% of works targeted physical design and manufacturing. In contrast, these areas accounted for only about 40% in 2016. Physical design and manufacturing work on geometrical representations of the chip, which can be represented as images. ML algorithms that work on images are most extensively explored and, hence,

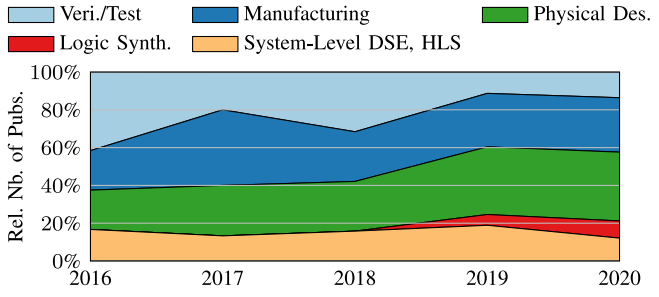

Fig. 2. Recent years show a trend towards physical design and lithography, reaching 65% in 2020 among all works that employ ML for design-time CAD.

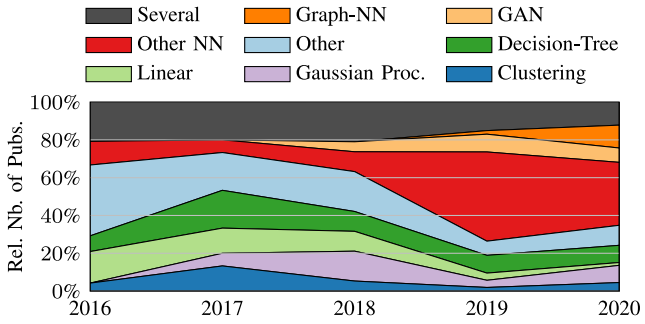

Fig. 3. Strong trend toward NN-based algorithms can be observed for design-time techniques. Recently, generative (GAN) approaches and Graph-NN gain more attention.

most advanced, which facilitates their usage. Early design steps, such as DSE, HLS, and logic synthesis, are often combinatorial problems and are relatively underrepresented. We conclude that physical design steps are well explored with ML and future research should focus on earlier design steps.

We next explore the range of ML algorithms applied to CAD. First, we study design-time steps. We divide the plethora of used algorithms into the categories listed in Fig. 3. Clustering algorithms are unsupervised algorithms that identify groups of examples based on a similarity metric. The most prominent algorithm of this group is k-means clustering [6]. Gaussian process models learn continuous functions based on prior knowledge (mean, variance, and covariance between samples) and observations [7]. Notably, Gaussian process models are capable to cope with noisy data and even inherently model noise. Linear models fit parameters of a linear kernel [6]. Decision-tree-based models represent knowledge as a tree, where every node represents a decision based on the features and threshold values that leads to a specific branch [6]. This category also includes ensemble models of decision trees, such as random forests or XGBoost. Neural networks (NN) consist of neurons that perform a linear transformation if their inputs followed by a nonlinear activation function [6]. Usually, neurons are aligned in layers. We extract two special types of NNs: 1) Graph-NNs [8] where the input is represented as a graph consisting of vertices and edges 2) and generative adversarial networks (GANs). GANs combine two NNs that are trained in a zero-sum game, where the generator learns to create realistic data, and the discriminator NN learns to distinguish generated from real data [9]. Some

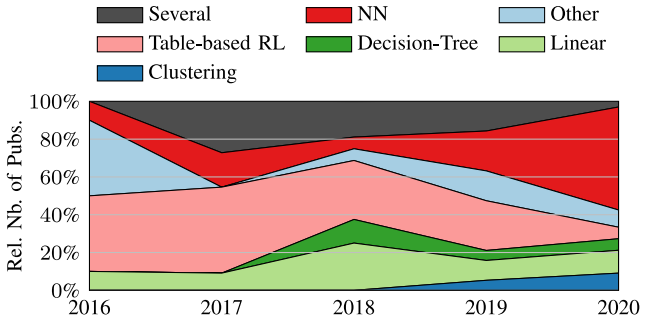

Fig. 4. Run-time techniques also show a strong trend toward NN-based algorithms. Table-based RL is the algorithm that has declined the strongest.

works implement several models. For instance, Fang *et al.* [10] and Zhou *et al.* [11] implemented and compare both classical and NN models for IR drop prediction and power prediction, respectively. We assign works that implement several types of models to a separate class.

Fig. 3 shows how the used algorithms in the design-time steps changed in recent years. Several trends are clearly visible. First, there is a strong trend toward NNs, accompanied by a decline in classical ML methods. The majority are feedforward networks (such as fully connected and convolutional neural networks) and recurrent networks (such as long-short-term memory (LSTM)), denoted "Other NN" in the figure. Generative approaches based on GANs also have been explored since 2018. 2020 has shown a trend toward Graph-NNs that exploit the graph-based representations of circuits for instance in logic and physical design steps. Finally, while in 2016, a significant fraction of works (about 20%) train and compare different ML algorithms for their problem, fewer works still do this in 2020. It appears that more works will use NNs in the near future, with an increasing use of Graph-NNs.

Fig. 4 shows the algorithms used for run-time management. These are mostly the same algorithms also used in design-time techniques. One exception is table-based RL, i.e., Q-learning [12]. Q-learning simply stores learned values in a lookup table. Many trends that we observed in design-time techniques are also valid here. NN are increasing, classical algorithms are declining. Unlike the design-time steps, GANs and Graph-NNs have not been used for run-time management. A large fraction of run-time works (45% in 2017) used table-based RL. This algorithm is strongly declining and replaced by deep RL (DRL), i.e., NN-based RL. The approach that several ML algorithms are implemented and compared is also decreasing in run-time techniques. It appears that NN-based techniques will account for the majority of techniques in the near future.

III. Patterns in Machine Learning for CAD

Approaches that employ ML for CAD can be classified according to three main criteria.

1) The problem type to be solved with ML: make predictions, suggest actions, and generate data.
2) The design step.
3) The ML algorithm.

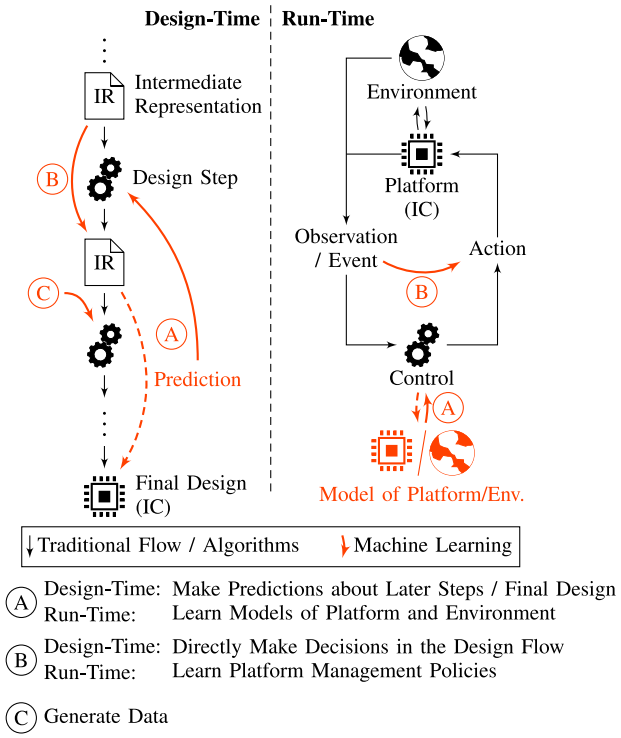

Fig. 5. Patterns how ML can be used for CAD: design-time (left) run-time management (right).

The three main alternatives of the first criterion, the problem type, are illustrated in Fig. 5. This section presents an overview of the type of problems and corresponding ML algorithms to lay the foundation for a detailed discussion of techniques in the next section.

A. Prediction of System Properties

The first pattern to employ ML for CAD is predicting properties of various aspects of the system: the design itself; the run-time platform; or the environment in which it operates. At design time, these can be properties arising in the following design steps (e.g., routing congestion) or properties of the final design (e.g., power, performance, area). At run time, these can be properties of the platform (e.g., power) or models of the environment (e.g., workload). The ML models are also sometimes called *surrogate models*. At both design time and run time, the output of the model is used in an optimization loop that explores the design space or action space.

Since the underlying mechanisms are very similar, the same ML algorithms are employed in design-time and run-time techniques. The employed algorithms belong to supervised learning, where training data are present in the form of input-output pairs of the model. The problem can be a regression problem (the outputs are continuous values) or a classification problem (the output is one out of a finite set of classes). There exist a plethora of different algorithms [6] ranging from simple linear regression models and tree-based models, to deep NNs [13]. Since these algorithms are most commonly known, we omit a detailed explanation here.

The output of such models contains little information as to *how* to optimize the design or run-time management. However, these models provide input to a traditional optimization algorithm that repeatedly calls the model. The repetitive use of these models means that maintaining a low inference overhead is key, limiting the complexity of employed models.

B. Decisions for Design-Time and Run-Time

The second pattern is to use ML models to directly make decisions in the design flow or run-time management: schedules, placements, v/f-level settings, etc. In contrast to Section III-A, where the ML model would, for example, answer the question "If this net would be routed here, what would be the implications?" such a technique would answer the question "Where should this net be routed?" The ML models replace the traditional methods.

This form of modeling can be tackled with both supervised and semisupervised algorithms. This can be for instance classifiers that classify between a discrete set of actions. Physical design and lithography are image-based design steps, where solutions can be expressed as images (e.g., routing path, lithographic mask). Therefore, inputs and outputs to the ML algorithm may be images. Convolutional autoencoders (AEs) are NNs that transform one image into another and, therefore, are well suited [6]. AE comprises two NNs: 1) an encoder and 2) a decoder. The encoder learns an efficient encoding of the input data to a lower dimensional latent space, whereas the decoder learns either to reconstruct the original data from the encoding or to transform the encoding to a target image. Simple classifiers and AEs are still trained in a supervised manner with a unique output for every input pattern. This is not always the case in CAD problems. Different solutions may have a very similar quality of result. In these cases, training ML model in a supervised manner requires unnecessary effort to learn *the single* solution represented in the training data instead of *any* good solution.

As a solution, RL-based techniques [12], [14] can be employed that let the ML agent take actions on the design, such as transforming a logic circuit. After every action, the RL agent is given a reward that reflects the current quality of solution. The goal of the agent is to maximize its long-term reward. The agent learns by exploring the potential actions and observing the reward. RL can easily cope with several actions leading to a similar quality of result. There are many different implementations of RL ranging from table-based Q-learning [12] to NN-based DRL [14]. RL-based techniques have the additional advantage that they perform online learning, which is especially useful for adaptive run-time management.

Finally, GANs have been proposed to circumvent the problem of nonunique model outputs [9]. As explained earlier, two NNs are used: 1) a *generator* and 2) a *discriminator*. The generator creates data from random noise, whereas the discriminator distinguishes generated from real data. Both NNs are trained alternately in a zero-sum game. Training the generator teaches it to create data that are indistinguishable from real data for the current discriminator. Analogously, the discriminator learns to detect generated data. By repeating this training cycle, both get better until, at some point, the generator is capable of creating deceptively real-looking data

TABLE I
COMMONLY OBSERVED PROBLEMS IN DESIGN-TIME CAD AND SUITABLE ALGORITHMS

Step	Problem	Algorithms
Common for all steps	Prediction of properties (power, etc.) Tool parameter tuning Directly perform optimization actions Augment incomplete solution	Linear regression, regression trees, NN, etc. RL, surrogate models (supervised learning) RL Conditional generative adversarial network (CGAN)
System-Level DSE and HLS	Steer exploration	Bayesian optimization, surrogate models
Logic Synthesis	Netlist optimization	Graph-NN
Physical Design	Netlist as an input Geometrical optimization	Graph-NN Convolutional NN (CNN, conv. CGAN, conv. AE, etc.)
Lithography and Manufacturing	Mask generation and optimization	Convolutional NN (CNN, conv. CGAN, conv. AE, etc.)
Testing	Anomaly detection	Clustering, Dimensionality reduction
Device / Technology Modeling	Physical modeling	Combination of domain knowledge with (small) ML models for fitting

without ever having seen real data. Conditional GAN (CGAN) is an extension of GAN where both generator and discriminator additionally are provided with partial information of data. The generator learns to reconstruct the missing parts, whereas the discriminator learns to distinguish reconstructed data from real data. Finally, the trained generator is employed for the CAD problem. An advantage of this approach over supervised learning is the capability to cope with nonunique solutions. This capability comes from not training the generator with concrete labels that it tries to reproduce, but instead training the generator with the help of the discriminator that can learn to classify several solutions as valid (real).

C. Data Generation

Some processes require a lot of data to be able to perform analyses. These data may be expensive to collect—either financially or timewise. There are two fundamental ways on how to generate data that follow the same underlying distribution as the training data. First, the underlying probability density function can be *explicitly* estimated (learned) [6], and new data can simply be drawn from it. However, such an approach works if correlation between different features is easy to capture, but fails if features show high and complex correlation, such as individual pixels in images. Therefore, recent algorithms only *implicitly* learn the data distribution. Examples are AEs, variational autoencoders (VAEs) [15], and GANs [9]. New data can be created with AE by adding a small perturbation to the encoding of a valid sample from the training data before decoding. However, such an approach may be limited to only creating data that are similar to individual training samples. VAE are extensions of the AE topology that enforces that the encodings use the full latent space in a continuous manner. Therefore, new data can be generated by passing random noise to the decoder. GANs also comprise two NNs. The generator is trained explicitly to create new valid data from noise, while the discriminator is trained to distinguish real from generated samples. The two NNs are mutually trained in a zero-sum game.

Creating new data is only required for design-time processes like early technology evaluation [16]. This approach is not employed in run-time techniques.

IV. RECENT WORK ON MACHINE LEARNING FOR DESIGN-TIME CAD

This section provides a summary of how ML can support the design phase of IC. It covers the system-level design, logic synthesis, physical design, device and circuit design (both analog and digital), and testing. Table I presents an overview of common problems in different steps of the design process. The details for each step are presented in the corresponding section.

A. System-Level DSE and HLS

System-level DSE and HLS are the first steps when designing IC from abstract specifications. The two are complementary. System-level DSE determines the overall architectural parameters, while HLS performs logic design.

System-level DSE determines the overall parameters of the design, e.g., cache sizes, processor core configurations of a multicore processor. Assessing individual configurations usually requires expensive simulations. Surrogate models can be used to replace simulations with a faster, yet less accurate approximation. An important property of such a model is to be able not only to rate a single configuration, but to actively steer the optimization toward the optimum. Mariani *et al.* [17] used Bayesian optimization as a surrogate model to speed up DSE of a multicore processor design. Every synthesized design accounts for one training example. Bayesian Optimization assumes a continuous objective function (e.g., power) with respect to the design parameters. Under this model, the objective function is accurately known close to an already synthesized configuration, and uncertainty increases with distance from a known configuration. Joardar *et al.* [18] used local search to find good parameters of a 3-D network-on-chip from a given starting point. They use ML (demonstrated with regression trees) to learn the objective function. The objective function is in turn used to find promising starting candidates. The main difference as compared to Mariani *et al.* [17] is the local optimization. Deshwal *et al.* [19] improved the scalability of DSE by learning simpler tree-based models to narrow down the design space toward the optimal configuration of a 3-D many-core processor. Powell *et al.* [20] predicted the

power and execution time of applications on FPGA soft processors. The application is represented by coarse statistics about instructions and memory accesses. This technique aims at speeding up early DSE of FPGA soft processors, which are defined by parameters such as cache organization, presence of floating-point hardware, and clock speed.

HLS is a form of logic design one level above register-transfer design. While register-transfer design requires the sequential behavior of the logic to be fully specified, HLS schedules operations to create a sequential machine; it also allocates operations to function units, a step that cannot be taken in a specification that does not bind operations to particular sequential time steps. Like in DSE, surrogate models can be used to obtain fast estimates of area, performance, or power. Liu and Carloni [21] employed a surrogate model to optimize HLS knobs, such as loop manipulations or array implementations. They present transductive experimental design to select a representative set of knob settings, which are used for initial training of the model. They then use the surrogate model to select the next candidate knob setting to iteratively refine the current best solution. Zhong *et al.* [22] also use a surrogate model to find a near-optimal set of HLS parallelism options. They put a strong focus on fast DSE, and parse and analyze a C/C++ software implementation of the kernel with different settings such as loop unrolling using LLVM without invoking HLS. These traces are fed into the ML model for area and performance estimations to find the best settings for an FPGA implementation. HLS is only performed once for the chosen settings. By using an abstract intermediate representation, this approach is capable of generalizing across many different designs. Zennaro *et al.* [23] learned the resource requirements of control register interfaces of regular SoC components when implemented on an FPGA. They describe the interface using high-level features such as the number of readable registers. Dai *et al.* [24] trained and compare several ML models to predict resource requirements from HLS reports. They also explore multitaks learning to exploit correlations between the target metrics.

Ustun *et al.* [25] predicted the circuit delay in HLS. They identify that the mapping of operations to hardened structures on an FPGA like DSPs significantly affects the delay. Such mapping mostly depends on local structures in the data-flow graph. They present a prediction technique based on Graph-NNs, which captures the local neighborhood of nodes (local structures) to predict the delay.

Another branch of work uses ML models to directly select the optimizations to perform. Chen and Shen [26] target the problem of scheduling in HLS for FPGAs. A model repeatedly selects a shift operation of a node of the data flow graph to earlier or later cycles based on the current schedule. They train the model once but also propose to use RL to further train the model during usage.

B. Logic Synthesis

Logic synthesis transforms the RTL description of a design to an optimized gate-level representation in the target technology. In this process, a number of transformations are applied to the design for logic optimization and minimization, mapping to entities of the target technology, and post-mapping optimization. These optimizations are performed on a representation of the design as a netlist, which is commonly represented as a graph of components and connections. Graph-NNs have been proposed recently to directly operate on graphs, allowing to directly make use of the underlying structure of connections.

The majority of works targeting logic synthesis use ML as surrogate models to estimate properties of the design. These estimations can be used to guide optimization. Zhang *et al.* [27] used a Graph-NN to propagate average toggle rates through combinational logic for power estimation. By operating on a per-gate granularity, they achieve generalization between different workloads and circuits. All these works only develop the estimation technique in isolation and do not perform any optimization. Pasandi *et al.* [28] predicted the error rate of an approximate circuit with the help of NN. They also develop an iterative optimization algorithm for power/area minimization based on the predicted error rates.

A smaller set of works uses ML to perform the optimization itself. Hosny *et al.* [29] used RL to perform logic optimization on And-Inverter graphs in order to minimize the area under a timing constraint. They represent the state of the network as a 7-D vector and select in each step one of seven possible actions. The state design allows reusing a policy trained on one design on another design. The RL agent is implemented using an actor–critic NN.

Finally, some works employ the classical synthesis tools and use ML at a higher level. Kwon *et al.* [30] used NN-based recommender system to tune parameters of the design flow (logic synthesis and physical design). They demonstrate that their approach generalizes to different technology nodes.

C. Physical Design

Physical design transforms a design from a graph-based representation (consisting of components and connections) after logic synthesis to a geometrical representation consisting of shapes of materials. Again, Graph-NNs are well suited to parse graph-based representations. The geometrical representation can be depicted as images. Computer vision (image classification and transformation) is a mature application for ML [4], [31], [32], hence developed algorithms for computer vision are prime candidates to be adapted to physical design, as well. Prominent examples are convolutional NN and variants thereof, such as convolutional CGAN, or convolutional AE. A typical physical design flow involves multiple stages, including placement of components on the die, routing of connections, clock network synthesis, and power/ground network synthesis. We describe these steps in the following.

1) Placement: Providing the significance of placement on chip, a mass of research achievements have been made in the past several decades. However, researchers are still not satisfied with the efficiency of previous chip placement algorithms. Due to the massive scale of modern designs, the placement process is usually complicated, tedious, and time consuming.

Mirhoseini *et al.* [33] proposed macro block placement as an RL problem and train an agent to place all macros of a

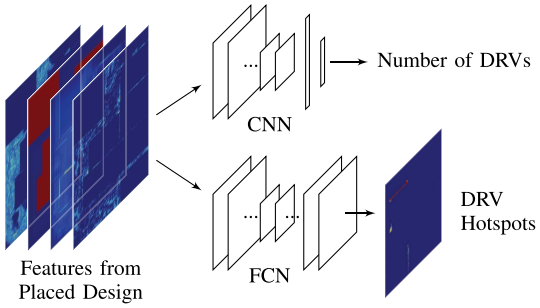

Fig. 6. Physical design has many similarities to image classification, which is the reason why CNNs are suitable models. RouteNet [36] proposes two CNNs, a conventional CNN to predict the number of DRVs for routability forecast, and a fully convolutional network (FCN) to predict the location of DRV hotspots. Figure is based on [36].

given chip onto the placement canvas. The general motivation for leveraging RL is to learn from past experience and improve the ability to place macros. Specifically, such an agent should be well trained over extensive chip blocks in order to gain as much experience as possible and further improve generalization ability. In fact, the deep RL approach does not require the agent to place nodes directly. Instead, at each step, the agent sequentially places the macros, and once all macros are placed, a force-directed method is applied to generate a rough placement of all standard cells. This RL-based approach outperforms state-of-the-art baselines and the produced results comparable to manual designs from experts. Liu *et al.* [34] used GAN to create noise maps from limited samples. These noise maps are fed into an optimization algorithm to find a placement for noise sensors. Barboza *et al.* [35] predicted the post-routing delay at the placement phase. They use handcrafted features in combination with classical ML algorithms like lasso regression or random forest to predict the delay of a single net without performing routing itself.

Placement has to be done considering the later routing steps, as routability is mainly influenced by the placement. To enable fast and accurate routability prediction, deep learning is introduced for its high performance in computer vision and other related tasks. Xie *et al.* [36] predicted the number and position of design rule violations (DRVs) after routing given a placed design before actually performing the routing (see Fig. 6). They exploit the similarity of the well-studied image classification problem to the 2-D placement, which also can be represented as an image. This allows them not only to use a similar model (CNN), but even to perform transfer learning from a different domain. CNN is pretrained on the ImageNet [3] dataset, which contains many photos of real-world objects, i.e., is unrelated to CAD, and then finetuned for the task at hand. The resulting predictions are used during placement to proactively avoid placements that are difficult to route. Tabrizi *et al.* [37] also supported the placement phase by predicting routing short violations given a placed design. They train NN but put a strong focus on feature engineering in contrast to [36], where raw images of the placed designs are fed to the NN.

It is an open challenge to automatically generate datapath-aware layout, since most conventional placers are designed to handle general-purpose placement and pay very little attention to such datapath layouts. However, some significant improvements have been made in the past few years. Ward *et al.* [38] proposed a new unified placement flow that simultaneously handles random logic and datapath standard cells. Specifically, graph-based and physical features are extracted from the netlist and fed into some effective classifiers (e.g., NNs) to classify the required datapath related patterns. Based on that, a datapath-aware placer, PADE, is proposed to handle datapath patterns and perform datapath-aware detailed placement.

Recently, some milestone studies have been proposed to maximize the use of GPU resources for accelerating global placement. Lin *et al.* [39] implemented the `DREAMPlace` placer to simulate the optimization of global placement as the NN training problem, so that it is able to leverage the widely adopted deep learning toolkit PyTorch with customized kernels and operators, and further make use of GPUs for extreme acceleration. `DREAMPlace` is designed based on the electrostatic-based placement algorithm, proposed by Lu *et al.* [40], which models the layout and netlist as an electrostatic system and attempts to find the balancing state with the lowest electrical energy via solving a *Possion's equation* by applying discrete cosine transformations. `DREAMPlace` can achieve over 30× speedup without quality degradation compared to state-of-the-art multithreaded placers.

Some works employ ML to select the tool parameters. Agnesina *et al.* [41] targeted FPGA place&route and built several models with the goal of accelerating compilation time. These models classify netlists into easy and hard classes, predict the best tool parameters, or predict compile time. This work uses stacked models that combine the output of various models with different algorithms by linear regression. Agnesina *et al.* [42] later targeted ASIC placement where they tune tool parameters with actor–critic deep RL. The state comprises the netlist and the current tool parameters. The netlist is represented both with handcrafted features and with learned encodings by using a Graph-NN, which both are passed to a multihead actor/critic NN. The goal (reward) is to reduce the wire length. Xie *et al.* [43] also automatically select tool parameters. Their technique starts with clustering-based sampling that exploits knowledge from prior designs to train a tree-based surrogate model, which is then further refined iteratively.

Lu *et al.* [44] targeted the problem of partitioning 3-D integrated circuits. They first perform conventional 2-D placement with relaxed constraints (smaller footprints) and then use clustering to assign nodes to 3-D layers. Similar to Agnesina *et al.* [42], they use a combination of handcrafted and learned features, which are obtained with a Graph-NN.

2) Clock Network Synthesis: Clock skew is the fundamental metric for estimating clock performance. It has been shown that modifying the latch placement locations is an effective method to reduce overall local clock tree capacitance, which affects the clock skew. At the same time, there were three prior latch placement modification techniques: 1) latch shifting; 2) latch clustering; and 3) latch banking. Since the packed latch cluster placements are produced in the previous physical design flows, Ward *et al.* [45] identified better solutions for

the technology library and provided the physical design flow a choice of templates to choose from.

Lu *et al.* [46] proposed GAN-CTS, which employs GAN and RL for clock tree prediction. They take flip flop distribution, clock net distribution, and trial routing results as input images. They also leverage a pretrained ResNet-50 on the ImageNet dataset and add fully connected layers for feature extraction. The framework utilizes CGAN to optimize the clock tree synthesis, of which the generator is supervised by the regression model. The policy gradient algorithm is leveraged for the RL-based clock tree synthesis optimization.

3) Routing: The routing step establishes physical connections between endpoints of the already placed devices that belong to the same signals. Yu and Zhang and Alawieh *et al.* applied a generative adversarial network (GAN) to learn the correlation between FPGA placement and routing congestion [47], [48]. Yu *et al.* [49] proposed a pin accessibility prediction model to refine the placement results. They propose to find the best spacing by brute-force search for the patterns between every two pins. Hung *et al.* [50] and Liang *et al.* [51] customized the network architectures for the prediction of DRV maps after global routing stage and placement stage, respectively.

The routing process is a very complicated and time-consuming task that would be difficult, if not impossible, to be solved by pure ML methodologies. Therefore, combining ML models and traditional algorithms is promising, such as that in [52], where a traditional algorithm is guided by the soft decisions made by ML models. In this way, better guarantees could be obtained given the soundness of the traditional algorithm. We also observe that, in the CAD flow, there usually exist different implementations of the same design that achieve similar performance. Therefore, supervised learning approaches are often infeasible, since there is hardly a one-to-one mapping from inputs to outputs. This is the reason for the wide usage of generative approaches (e.g., with GANs), which account for such degrees of freedom in the implementation.

In addition to congestion prediction, Qu *et al.* [53] observed that the order of nets to be routed in a sequential router [54] can significantly impact the routing quality, especially the number of DRC violations. They propose RL-based algorithm to learn the ordering policy that minimizes the DRC violations from the net features. While in RL, each input design is regarded as one distinct environment, they customize the network architecture of the RL agent such that it is applicable to different designs.

Not only the classical design flow can benefit from ML but also security measures like split manufacturing can be attacked. Li *et al.* [55] and Zeng *et al.* [56] aimed at reconstructing higher metal layer connections from full information about the lower layers. Both techniques predict the likelihood that two pins are connected by the higher metal layers and thereby help in reconstructing the whole chip.

4) Power/Ground Network: Power delivery network (PDN) design is a complex iterative optimization task, which strongly influences the performance, area, and cost of a chip. To reduce the design time, recent studies have paid attention to ML-based IR drop estimation, a time-consuming subtask. Previous work usually adopts simulation-based IR analysis, which is challenged by the increasing complexity of chip design. IR drop can be divided into two categories: 1) static and 2) dynamic. Static IR drop is mainly caused by metal wire resistance in the power grid, while dynamic IR drop is caused by signal switchings and local current fluctuations. In IncPIRD [57], the authors employ XGBoost to conduct incremental prediction of static IR drop, specifically, the IR value changes caused by the modification of the floorplan. For dynamic IR drop estimation, Xie *et al.* [58] proposed PowerNet, which aims to predict the IR drop values of different locations and models IR drop estimation as a regression problem. This work introduces a "maximum CNN" algorithm to solve the problem. Besides, PowerNet is designed to be transferable to new designs, while most previous studies train models for specific designs. A recent work [59] proposes an electromigration-induced IR drop analysis framework based on CGAN. The framework regards the time and selected electrical features as input images and outputs the voltage map. Another recent work [60] focuses on PDN synthesis in floorplan and placement steps. This article designs a library of stitchable templates to represent the power grid in different layers. In the training phase, simulated annealing is adopted to choose a template. In the inference phase, fully connected NN and CNN are used to choose the template for floorplan and placement steps, respectively. Cao *et al.* [61] trained several ML models to predict the quality of the power delivery network (bump inductance) in order to fill the gap between inaccurate, but fast estimation tools and accurate, but slow signoff tools.

D. Analog Physical Design

Physical design of analog circuits is considerably more complicated than design of digital circuits because there exists more diverse set of constraints that need to be satisfied. In addition, performance control at the analog physical design level is extremely challenging. As a result, automated design of analog circuits is not as mature as its digital counterpart. Nevertheless, ML has been employed also for analog design [62], [63].

Chen *et al.* [63] proposed an analog physical design framework, whose main steps comprise parametric device generation, layout constraint extraction, placement, and routing. The constraint extraction searches for symmetries in the circuit at different abstraction layers. Wang *et al.* [64] proposed a customized Graph-NN approach for analog placement performance prediction, which helps analog IC placer to obtain a solution similar to manual designs. Zhu *et al.* [52] used a VAE to learn the probability that an analog net is routed in certain areas. The resulting probability heatmap guides a heuristic routing algorithm, which guarantees that certain requirements like symmetry for specific nets are fulfilled. Training is performed using human-routed designs, i.e., they learn from human designers. Xu *et al.* [65] used a CGAN to create well regions in analog designs. The generator is trained to augment placed designs with well regions, while the discriminator is trained to distinguish machine-generated and human-generated well regions. Therefore, similar to the

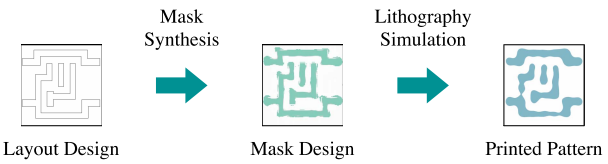

Fig. 7. Typical mask synthesis and verification flow.

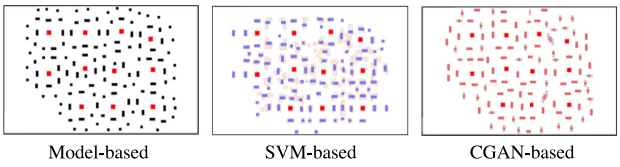

Fig. 8. Comparison of SRAF results between model-based, SVM-based, and CGAN-based approaches [70].

previous approach, this approach learns from human designers. Li *et al.* [66] proposed a transferable automatic transistor sizing method leveraging both Graph-NN and RL. Benefiting from the transferability of RL, they transfer the knowledge between different technology nodes and even different topologies. Meanwhile, Graph-NN is utilized to involve topology information into the RL agent.

Shook *et al.* [67] predicted the parasitics (interconnect resistance and capacitance) at the prelayout step. They extract features such as the number of connections for each net and use a random forest regression model to estimate its parasitics. Ren *et al.* [68] target the problem of predicting layout parasitics and device parameters and train a Graph-NN that exploits local structure in the netlist. They train several models in a hierarchical way to cope with the large range of parasitics values. Hakhamaneshi *et al.* [69] described an evolutionary algorithm with deep learning to tune parameters of an analog design. NN-based discriminator is used to prefilter new specimen of the population without having to perform expensive SPICE simulations. The NN compares two specimen and outputs probabilities that the first design outperforms the second one w.r.t. different metrics. The inherent symmetry of this problem is reflected by the network design (i.e., by restricting weights).

E. Lithography and Manufacturing

In modern VLSI manufacturing, lithography plays a critical role, as it determines the printing resolution and robustness of the manufacturing process. Fig. 7 shows a typical flow for mask synthesis and verification, consisting of mask synthesis and lithography simulation. Mask synthesis takes a layout design as input and produces the mask design with better printability. Lithography simulation then takes the mask design as input and computes the printed pattern with lithography models. Since mask designs can be naturally represented as images, ML techniques, such as CNN, are suitable to tackle lithography problems like mask synthesis, lithography modeling, and lithography hotspot detection. In addition, we also cover ML applications in other manufacturing tasks like yield estimation.

1) Mask Synthesis: Mask synthesis typically contains inverse lithography optimization steps, such as SRAFs generation and optical proximity corrections (OPCs). In SRAF generation, small rectangular features are inserted into the mask to assist the patterning of target features. These features are too small to be actually printed, but they can improve the patterning robustness of the target ones. In OPC, the edge segments of target features are adjusted for light compensation. The quality of mask synthesis is usually measured with two metrics: 1) edge displacement errors (EPEs) and 2) process variation bands (PVBands).

The early attempt of ML for SRAF generation formulates the problem into a classification task [71]. By dividing the mask into pixels, Xu *et al.* proposed to use logistic regression and support-vector machine (SVM) to predict the probability of SRAF being present at each pixel. They demonstrate comparable EPE and PVBand with 3×–10× speedup on a 10 μm× 10 μm mask clip compared with model-based approaches in a commercial tool [72]. The drawbacks of such an approach include manual feature engineering, high prediction complexity, and lack of global view in prediction, as each pixel is treated separately and we need to make predictions for each pixel. To overcome such drawbacks, Alawieh *et al.* [70] formulated the problem into an image-to-image translation task that tries to obtain the entire solution with one prediction and a legalization step. They propose a multichannel heatmap encoding method to handle the SRAF size rules and leverage the CGAN model to predict the SRAF results. Eventually, they can achieve 144× speedup and closer PVBand compared with the model-based approach [72]. Fig. 8 compares the solution generated from the model-based approach (golden) [72], SVM-based approach [71], and CGAN-based approach [70]. We can see that the CGAN-based approach matches the golden result much better globally.

While OPC can also be formulated as an image-to-image translation task, the problem becomes more complicated to manipulate the edge segments of features. Yang *et al.* [73] proposed a GAN-OPC framework to generate the initial OPC solution. They develop the generator as an AE and the discriminator judges the quality of the generated mask, as shown in Fig. 9. To bootstrap the training, the discriminator is initially replaced with lithography simulation. Unlike SRAF generation where we can obtain the final solution with a simple legalization step, the initial solution can only serve as the starting point for conventional OPC iterations. It eventually can achieve similar solution quality with half of the conventional iterations, i.e., around 2× speedup over a conventional gradient-based OPC solver [74]. Recently, Jiang *et al.* [75] proposed to replace the backbone of the conventional OPC solver with NN and GPU-accelerated lithography simulation kernels. In this way, they can achieve 70× speedup with even better solution quality than the conventional solver [74].

Most the previous studies focus on mask clips and require to sweep the layout for full-chip mask synthesis. Chen *et al.* [76] proposed a full-chip mask optimization engine by multistage clustering and clip generation. They demonstrate 5× speedup and better solution quality even compared with the commercial tool [72].

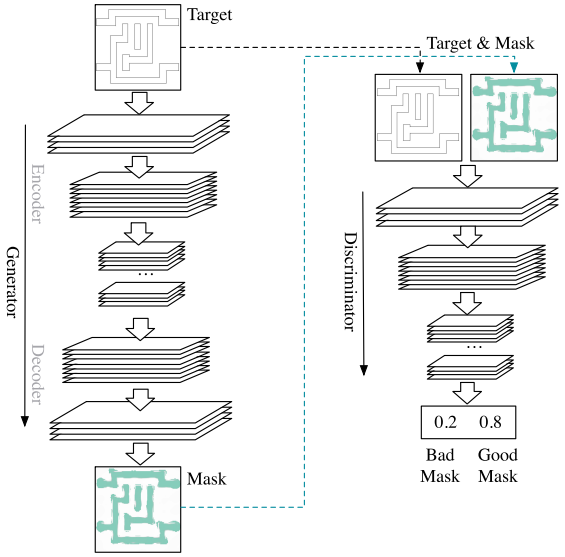

Fig. 9. Generative techniques allow to cope with ambiguity in the design process. Many different masks may result in similar quality. GAN-OPC [73] uses two NNs following the GAN principle. The generator creates masks, whereas the discriminator rates the quality of these masks.

2) Lithography Modeling: Lithography modeling is a step enabling lithography simulation of printed patterns given mask clips. It is not simply a step next to mask synthesis, as shown in Fig. 7. In practice, lithography simulation is a subroutine iteratively called during mask synthesis to verify the printing quality of masks. Lithography modeling consists of optical modeling and resist modeling. The former computes the light intensity map (aerial image), and the latter simulates the slicing thresholds for patterning and determines the printed shapes.

Watanabe *et al.* [77] discovered that ML-based resist models have the potential to outperform conventional compact models [72] in accuracy and achieve much higher efficiency compared with rigorous simulation [78]. They formulate the resist modeling problem into a regression task and develop CNN models to predict the slicing thresholds on the aerial image. The printed patterns can be computed with the slicing thresholds and the aerial image. Since the printed patterns can also be viewed as an image, Ye *et al.* [79] further formulated the entire lithography modeling problem on contact layers into an image-to-image translation task and develop CGAN+CNN framework for end-to-end modeling. The CGAN learns the shapes of patterns and the CNN learns the locations. In this way, they can achieve less than 1 nm average edge displacement error with more than 1800× speedup over rigorous simulation [78]. Recently, they further investigate the 3-D structure of masks by considering the mask topography effects, and formulate a multidomain image translation task to predict 3-D aerial images [80].

To obtain accurate models, a large amount of labeled data for training is required, which is often difficult to obtain. To enable few-data learning, Lin *et al.* [81] proposed to leverage transfer learning and active data selection to reduce the amount of training data. They utilize the data from old technology nodes to build an initial CNN model and finetune with a few labeled data from the target node. When selecting the data from the target node, they perform K-Medoids clustering to the features and choose the cluster centers to query their labels and form the training dataset. In this way, 3×–10× reduction on the amount of data samples from the target node can be achieved within an industrial-strength range of prediction errors.

3) Lithography Hotspot Detection: Different from lithography modeling that simulates the printed patterns with optimized masks, lithography hotspot detection aims at early detection of layout patterns that may cause printing failure such as short or open. This problem is usually formulated into a binary classification task taking a mask clip and determining whether it contains hotspot patterns. The key challenges include high image resolution and data imbalance, as most of the patterns are nonhotspots and hotspot patterns usually only take a few percentage. Thus, it is a biased learning task and we shoot for maximum prediction accuracy and minimum false alarms.

Shin and Lee [82] proposed to use CNN to predict the hotspot probabilities and augment the training data by flipping the mask clips. Yang *et al.* [83], [84] developed a dedicated discrete cosine transformation (DCT)-based feature representation to reduce the mask image by omitting the high-frequency components with custom CNN structures. They also suggest a biased learning procedure to finetune the models taking advantage of the ReLU property.

Despite the following-up studies to further improve the model accuracy [85]–[87], research has been conducted to investigate data-efficient learning under various scenarios [88]–[91]. That is, improve the accuracy with as few training data as possible. For instance, Ye *et al.* [89] and Yang *et al.* [90] introduced active learning to reduce the label querying overhead by examining the prediction confidence of models. They assume there are a pool of unlabeled data samples whose labels can be queried at certain costs. Then, they can gradually improve the model accuracy by selectively adding samples into the training dataset with minmum costs. Chen *et al.* [91] considered the scenario where a pool of labeled samples for training and unlabeled samples for testing are available, but there is no freedom to query for the labels of new samples. They propose to leverage the distribution of unlabeled samples to improve the generality of the model and introduce a self-paced semisupervised learning technique for few-data learning.

Recent study further reveals that clip-based hotspot detection may require numerous predictions when it comes to a full-chip scale. Thus, Chen *et al.* [92] reformulated the problem into an object detection task given arbitrarily sized mask regions. They introduce a clip proposal network consisting of a regression branch to predict the clip sizes and locations, and a classification branch to detect whether the clip contains a hotspot. They demonstrate 50× speedup over clip-based hotspot detection [84] with even higher accuracy and lower false alarms at full-chip scales.

4) Yield Estimation: ML can also help with yield estimation and analysis. Ciccazzo *et al.* [93] built an SVM-based surrogate model to estimate the yield for given design parameters.

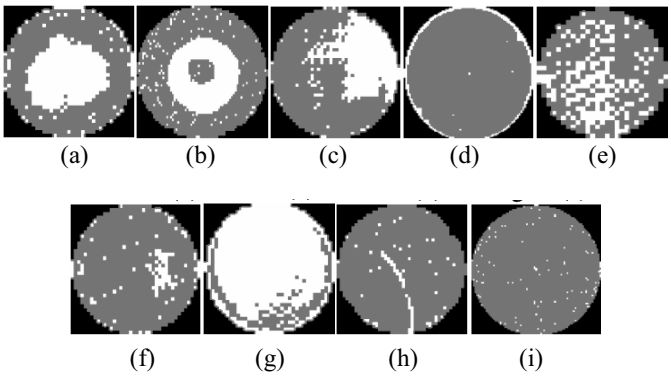

Fig. 10. Wafer samples of different defect types [97]. (a) Center. (b) Donut. (c) Edge-Loc. (d) Edge-Ring. (e) Random. (f) Location. (g) Near-Full. (h) Scratch. (i) None.

The surrogate model can then be used to speed up a heuristic yield optimization technique.

There are also studies on systematic failure and defect patterns for yield analysis [94]–[97]. Nakata *et al.* [94] helped engineers finding the cause of failure (defective manufacturing devices) from wafer failure map patterns and manufacturing histories. They employ several ML algorithms. K-means clustering is used to find groups of similar failures, a pattern mining algorithm finds defective manufacturing devices, and NN for recurrence monitoring, where the NN classifies known failures. Alawieh *et al.* [97] studied the wafer defect categorization problem. Given the locations of passing and failing dies on a wafer, the problem is to classify the wafer defect types like center or scratch, as shown in Fig. 10. They formulate the problem into an image classification task and propose a selective learning scheme leveraging CNN and an integrated reject option to maximize the prediction coverage and minimize misclassification risk. The selective scheme employs a pair of models (f, g), where f is the prediction model and g is the selection model serving as the binary qualifier for f. When $g = 0$, the selective model chooses to abstain from prediction. In this way, a tradeoff between misclassification risk and prediction coverage can be achieved. To tackle the data imbalance issue, they also propose a data augmentation technique with AE to create synthetic samples from underrepresented classes. Such a kind of wafer defect detectors can guide process engineers for yield optimization.

F. Verification and Testing

Errors that prevent the design from adhering to its specifications may happen at every design step. The earlier such an error is detected and fixed, the lower are the induced delays and costs of the design process. Therefore, early detection is indispensable to avoid design iterations and keep the design process economical. As a result, verification and testing of the design is performed after each step of the design and manufacturing process.

Mostly, verification is performed using simulations [98]. The design is exercised with input stimuli and its outputs are compared to golden outputs. Errors can only be detected if a high *coverage* is reached, i.e., the fraction of functions exercised in the test. High coverage can only be achieved by many simulations with various stimuli. Two challenges arise from this. First, the required simulation time is high, and second, creating stimuli to achieve a high coverage is difficult.

ML has been employed to both these challenges. Toward speeding up simulations, Li and Jacob [99] modeled the DRAM access latency. They classify incoming requests based on features about this and previous requests. These classes coarsely correspond to DRAM states (e.g., row hit/miss). Each of the classes is assigned with an average latency that is used as an estimate for the request at hand. As the focus of this technique is to speed up simulations, they use lightweight algorithms like decision trees. Lee and Gerstlauer [100] estimated power waveforms of hardware accelerators at different levels of knowledge about internals about the implementation. Cycle-level, block-level, or invocation-level models are built accordingly. Finally, Chen *et al.* [101] tackled the problem of increasing the test coverage. They use unsupervised learning to detect additional test points that improve an incomplete test plan. Ma *et al.* [102] targeted the problem of test point insertion, where a minimal number of observation points is added while maximizing fault coverage. They formulate the problem as a graph operation, where nodes in the netlist should be classified as suitable/nonsuitable for test point insertion. This problem is tackled with a multistage Graph-NN, which can cope with large imbalance in the classification problem.

After manufacturing, simulation is no longer required, as the manufactured device can be tested directly. Works that improve the coverage still apply. However, after manufacturing not one but many instances of the device exist with ideally identical behavior. This forms an opportunity to detect faulty devices by looking for outliers using anomaly detection algorithms. Kim *et al.* [103] used anomaly detection after every manufacturing process step to detect faulty wafers. They compare different dimensionality reduction methods and anomaly detection methods. DeOrio *et al.* [104] also used anomaly detection to detect the timestamp and signals involved in intermittent failures during post-silicon validation. They build clusters based on features from subsequent correct executions. Erroneous executions are then classified cycle-by-cycle to detect the timestamp and signals. This technique does not require generalization across different designs, but work by comparing many instances of the same design.

Analog designs have the property that inputs and outputs are continuous. This allows to search in a continuous space for the worst case operating conditions w.r.t. certain specified properties like common mode suppression. Hu *et al.* [105] used Bayesian Optimization to identify the worst case impact of manufacturing variability on the properties of an analog circuit. Bayesian Optimization iteratively builds up a new model from scratch for every design that guides the testing process toward the worst case operating point.

G. Device and Technology Development

Employing ML for technology development and to generate models of transistors and circuits have been initially

started in 1996, when Meijer investigated the possibilities of NNs for circuit simulations [106]. One of their objectives was to replace physics-based models, in which obtaining the needed configuration parameters (geometry, dopant concentrations, etc.) is typically very challenging, time-intensive and requires sensitive information from the semiconductor manufacturers. Additionally, their goal was to reduce the complexity of the models, which, in turn, reduces the run-time of the circuit simulations. The key limitation of the work, was the ML technology and computational power available at that time, as stated by: "Some behavior is beyond the representational bounds of our present feedforward NNs" [106]. To overcome that challenge, domain knowledge was used to enhance NNs (e.g., optimizing parameters and reducing complexity to single equations). A combination of domain knowledge and general ML techniques allows to build models that conform known physical dependencies, while still being flexible.

A recent approach in modeling transistors with NNs is from Zhang *et al.* [107] which aimed at assisting designing new technologies. Their work is on the material level where transistor details like geometry and dopant concentrations and other properties of the transistor are considered as a part of the input of their NN. With that framework, the work predicts the characteristics of new transistors in emerging technologies or different material properties.

An interesting approach was made by Lamamra and Berrah [108] who used a genetic algorithm in addition to the NN. This enhancement alters the training phase so that not only the internal parameters of the NN (i.e., the weights) are changed but additionally the structure of the NN (topology, number of nodes, etc.). This structural change of the NN enables the authors to obtain a model that minimizes the error further than just regular training. Unfortunately, the authors tested their framework only on one simple MOSFET transistor and only inferred the drain source current.

Another approach comes from Zhang and Chan [109] where methods are presented to develop a transistor model with NN and to minimize the needed data set for the training. The used NN is quite small (less than 15 nodes). In order to get satisfying results with the small network, authors heavily optimized the NN by employing domain knowledge to develop the network. For example, each node was connected manually to consider the physical dependencies between the input parameters and the electrical behavior of a transistor. This domain knowledge also resulted in the existence of specific nodes, carefully chosen to model the subthreshold current and nodes solely modeling the current above the threshold voltage. Additionally, the authors applied preprocessing to scale the input parameters. To reduce the minimum needed data, they used a sparse nonuniform data set. The approach does not model the temperature dependencies, but use more architecture-dependent parameters. A disadvantage of this small optimized network is the adaptability to new parameters and dependencies. Adding new parameters (such as ferroelectric parameters in recent emerging technologies) results in a need to adapt the layout of the NN with adding additional nodes in the hidden layers. This is in contrast to larger, more generic NNs, which can generalize such properties.

A similar approach to the work from Zhang *et al.* is the approach from Li *et al.* [110] which also tries to build the NN based on the device physics. The employed NN is tiny with less than ten nodes. A difference to other approaches is the use of different activation functions for different nodes. The small number of nodes enables the authors to train the NN with up to 5 000 000 epochs.

When it comes to emerging technologies, in which physics-based models are not fully developed or even available, ML can play a major role to replace traditional modeling and provide accurate estimations based on "learning from available measurement data." Recently, Klemme *et al.* [111] employed ML to model the negative capacitance field-effective transistor (NCFET), demonstrating the ability to predict with a high accuracy (>90%) the behavior of steep-slope transistors. They show that ML can be even employed to replace the standard cell library characterization. This enabled for the first time fast predictions of how changes in the underlying technology can impact the figure of merits of circuits. Hence, DSE to determine the best configuration for the ferroelectric material has become, as a result, possible. Such a DSE can provide guidelines to material scientists on how the different material parameters in their emerging technology should be tuned toward maximizing the efficiency of the circuit [112].

V. ML for Run-Time Management

This section gives an overview of ML-based techniques that support run-time management of ICs. The criteria are that inference is performed at run time and that the ML model is used to optimize the physical characteristics of computing platform operation, such as power, performance, or reliability. As explained earlier, we explicitly do not consider techniques that use ML to solve a user task (such as stroke detection), or that improve the performance of the inference itself, e.g., ML accelerators). Training can either be performed at design time, at run time, or a combination of the design-time training and run-time refinement. We consider two categories of run-time management: 1) those that directly learn platform policies to manage platform characteristics and 2) those that estimate characteristics of the computing platform and/or its environment for use by other management techniques. Table II gives an overview of common problems and suitable algorithms.

A. Learn Platform Management Policies

The first category of techniques directly learns policies that manage platform operation—for example, learning power management policies. The two most important methods for policy learning are RL and imitation learning (IL).

RL relies on defining the objective in the form of a single scalar value, the reward signal. Table-based centralized Q-learning is the simplest form of RL. Ebi *et al.* [113] reduced thermal gradients across a multicore processor and increase the performance through dynamic voltage and frequency scaling (DVFS). Shen *et al.* [114] performed power management with Q-learning. They accelerate training convergence by updating several virtual state transitions for each real state transition by exploiting system knowledge. Additionally, they build a

TABLE II
COMMONLY OBSERVED PROBLEMS IN RUN-TIME MANAGEMENT AND SUITABLE ALGORITHMS

Problem	Algorithms
Simple action learning	RL (table-based, NN), IL (linear, decision tree, NN)
Large state space	Deep RL, IL (linear, decision tree, NN)
Large action space	Distributed RL
Estimate current platform / environment state	Linear regression, regression trees, NN, etc.
Time series forecasting	Recurrent NN, ARMA (and variants)
Stochastic input / system behavior	MDP

high-level controller to determine the desired tradeoff between power and performance, which internally relies on the NN model to predict the power consumption. Shafik *et al.* [115] used *Q*-learning for DVFS. The learned policy is not intended to generalize across different workloads. Instead, workload changes are detected and the policy is updated through re-exploration. They use a small table (coarse quantization) to speed up the (repeated) exploration. Kim *et al.* [116] improved the lifetime of a processor using DVFS and power gating. Dinakarrao *et al.* [117] used *Q*-learning for temperature and reliability management of a multicore processor. Gupta *et al.* [118] used RL to decide the number of active cores and their v/f-levels in a heterogeneous multicore processor in order to minimize the energy consumption. They use NN-based deep *Q*-learning (DQL) to manage the large state and action space.

Even with DQL, state and action spaces might become too large if the number of cores gets too large, which reduces or prohibits convergence of the policy. In such a case, it is beneficial to split the centralized agent into many distributed agents. However, global convergence and cooperativeness between agents may be difficult to achieve. Chen and Marculescu [119] maximized the performance under a global power budget. The individual agents are coordinated using a global heuristic power budget reallocation algorithm. Li *et al.* [120] managed power states and control voltage regulators to minimize the energy-delay product (EDP). In their case, each agent is able to operate independently, since there is no global joint constraint. The techniques described so far use value-learning. Mao *et al.* [121] used policy-based RL to decide when to schedule jobs with the goal of maximizing the performance. They penalize the agent (negative reward) for every job in the system, which indirectly rewards finishing a job.

While it is intuitive to learn the direct actions to take (e.g., v/f-levels), this ignores that many heuristic algorithms perform well in certain scenarios. Ul Islam and Lin [122] used RL to switch between a set of preimplemented heuristic control policies at run time based on the workload.

RL-based techniques may suffer from high storage or computational overhead. This comes from the fact that learning is performed continuously at run time and all information required to continue learning must be retained. In contrast, IL learns an optimal control policy at design time, based on training data that captures a set of system states. IL reduces run-time overhead at the cost of adaptability. Park *et al.* [123] trained classical ML models to control the v/f-settings of CPU and GPU for performance and energy optimization. Training examples from the optimal control policy are created at design time by brute-forcing all possible settings for different execution phases. Mandal *et al.* [124] decided the number of active cores and their v/f-settings in order to minimize the energy consumption (with/without performance constraint). Kim *et al.* [125] controlled v/f-levels of v/f islands. They compare their IL-based technique to RL-based technique and demonstrate a significantly lower overhead.

All these IL-based techniques employ simple models such as decision trees to keep the run-time overhead low. Furthermore, all techniques use the *DAgger* algorithm [126] to make the learned policy more robust toward recovering from suboptimal decisions and unexpected system behavior.

The main advantage of learning actions directly is the possibility to abstract from detailed system behavior and focus on the design objectives instead. While this seems straightforward, there are several pitfalls. With IL, an optimal policy has to be generated. In easy cases, brute-force trying all possible actions works, but more complicated cases require heuristics, as well. With RL, defining the reward function is a challenge on its own. If the reward is not defined carefully, the agent might find a policy that results in high rewards but does not reflect the goal that the designer intended [127]. This effect is known as *reward hacking* [128].

B. Estimate Platform and Environment Properties

An alternative approach is to train ML model to estimate physical properties of the computing platform and the environment in which it operates. The models are used to predict future system changes, or to predict the impact of management actions before executing them. The results of this estimation are presumably fed to other subsystems, either automatic or manual, for use in system management.

Several studies develop methods to reconstruct the current, partially observable system state. Analog sensors for power or temperature are costly to implement and, therefore, usually are rare with only a few sensors per chip. Bircher *et al.* [129] proposed a simple linear model to estimate the current processor power from performance counter readings. Sagi *et al.* [130] augmented a linear model for processor power estimation with nonlinear transformations of the features to achieve a higher accuracy while maintaining a low overhead. Sadiqbatcha *et al.* [131] used a recurrent NN to estimate the current temperature of thermal hotspots at run time from processor performance counter measurements.

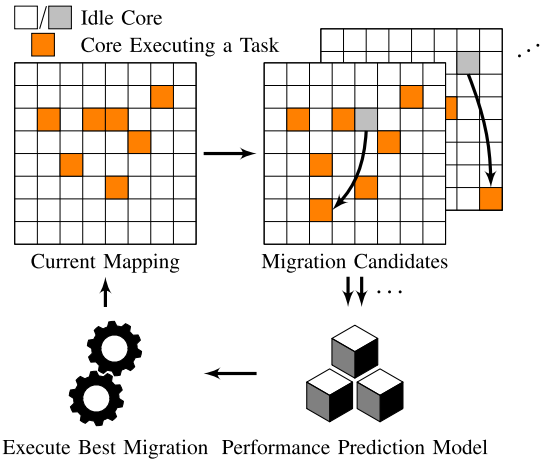

Fig. 11. Prediction-based resource management (here: task migration) selects the next action by predicting the impact of many candidate actions [134].

Kim *et al.* [132] presented an automated technique to create run-time power models for arbitrary integrated circuits that first selects signals to be monitored by clustering and then trains a regression model for run-time power inference based on the monitored switching activity. Rapp *et al.* [133] presented a processor boosting technique that internally builds on top of the NN model to estimate the sensitivity of the power and performance of running applications on v/f-level changes. They exploit the relation of these two metrics by building a multitask NN that simultaneously estimates them from performance counters.

While reconstructing the current system state already gives important information about the system to the control algorithm, predictions about the impact of potential actions are more meaningful. The majority of work has therefore focused on such problems. Fig. 11 visualizes this approach using an example of task migration [134]. The control loop traverses three phases. First, many action candidates (here: task migrations) are created. Then, the impact of each of these action candidates is predicted (here: performance after the migration). Finally, the action with the best predicted outcome is executed. The future state of system metrics such as power or performance depends on both the selected actions and the future characteristics of the workload and environment. To avoid the challenge of predicting workload and environment, many techniques predict how the system metrics would be now if another action would have been selected. Selecting the next action based on such predictions inherently assumes that the workload and the environment will not change within the next control step. While this is a strong assumption, it holds as long as control steps are short enough.

Gupta *et al.* [135] learned a linear model at run time that predicts the frequency sensitivity of workloads. They put a strong emphasis on adaptive learning rate (adaptive forgetting factor) to be able to quickly adapt to workload changes. Kim *et al.* [136] take traces from an application that is running on a certain core. They use NN to predict the power and performance of this application if it would be executed on another core with different microarchitecture. They use a cascaded NN to learn individual problems separately: change of performance counters, impact of the frequency, impact of the microarchitecture. Rapp *et al.* [134] predicted the performance of an application if it would be executed on another core in a thermally constrained many-core processor with heterogeneous last-level cache (LLC) access.

For some metrics, such as temperature, it is not reasonable to assume that they do not change within the next control step. Therefore, predictions about temperature always target future time steps. Zhang *et al.* [137] predicted how the temperature of a processor will behave if a certain application is started. They use application characteristics, as well as CPU-specific features. Abad and Soleimani [138] used NN to predict the temperature in the next seconds in a multicore processor based on information about the workload, as well as information about the frequency and cooling fan speed. Sagi *et al.* [139] predicted future power due to workload changes with LSTM NN.

The techniques discussed so far model properties of the platform and environment in a deterministic manner. However, especially in stochastic environments, it is important to model such stochastic behavior. Markov decision processs (MDPs) can be used to adapt operational characteristics of computing systems. A great deal of work has concentrated on the use of MDPs to model channel characteristics in communication systems. Li *et al.* [140] developed a framework for the design of digital predistortion systems that optimize the communication based on the MDP models created at design time. The group also used hierarchical MDPs [141] to efficiently model both the communication environment and the computing system platform. Bhuiyan *et al.* [142] described a probabilistic approach to energy-optimized scheduling of multicriticality systems. Their multicriticality model includes two criticality levels: 1) `HI` and 2) `LO`. Each task is guaranteed to execute to completion; processor mode switches are performed as necessary to increase clock speed so as to allow all tasks to complete their worst case execution. They characterize the execution time of each task using an empirical cumulative distribution function derived from a set of measurements or simulations. They iteratively solve for a minimum speed s_{LO} that guarantees schedulability along with a minimum-energy static schedule.

VI. Open Challenges

This section discusses open challenges when employing ML for CAD, as well as promising directions on how to solve them. Some challenges arise from the ML algorithms, some arise from the existing constraints in the CAD process.

A. Combinatorial Optimization Problems

ML can be incorporated into combinatorial problems either by approximating some heavy computation via surrogate models, or by acquiring better heuristics to solve a problem [143]. Combinatorial problems are often theoretically hard to solve, and ML approaches do not give any guarantee in terms of *optimality*. That is, we can never easily know how far away the output solution is from an optimal solution.

Despite the optimality issue, even generating a *feasible* solution itself is not trivial. Especially, when using neural networks, which are trained with gradient methods, it is important to carefully design operations that are differentiable to keep the whole model end-to-end trainable. As examples, pointer networks [144], attention layers [145], and sinkhorn layers [146] are complicated mechanisms for a neural network to output a permutation. In practical problems, there could be more cumbersome rules and constraints to be satisfied, which greatly brings about the difficulty in ML algorithm design.

Another issue is that many useful ML techniques are not established for combinatorial problems. One of the reasons might be due to the importance of introducing prior knowledge to the ML approach. For instance, it is commonly believed that CNNs can better extract useful features from image data than other NN models. Therefore, designing dedicated models for combinatorial problems might be critical to boost the performance.

Finally, scalability is a great challenge. Current methods usually experience performance degradation when applied to problems of larger size than what was used in training. It seems that using larger models and training on larger instances are the way to go, but it is at the cost of higher computational efforts. More importantly, it is nearly impossible to know *a priori* that how complex the model or how large the training samples should be, because we do not know the exact problem we are trying to solve [143] (i.e., the true data distribution).

B. Employ in Practice

There are some challenges involved with ML techniques managing the leap from research to employing them practice. The first challenge concerns the CAD flow. The existing CAD flow and corresponding tools have been developed and established in a process lasting several decades. It is mostly seen as rigid and immutable. As a result, techniques that do not fit the classical tool flow are less likely to be successful in the CAD community. This makes sense from the point of view that lots of optimization has been put into the existing flow and reinventing it may be a waste of effort. Instead, techniques are mostly developed as drop-in solutions to replace or enhance existing algorithms and tools. However, this also forms a limitation that potentially unnecessarily restricts new techniques and may capture the CAD flow itself in a local optimum.

A second challenge arises from intellectual property (IP) rights and licenses. Most functionality of a modern chip comes from licensed IP packages that need to be bought first. The most common types are hard IP, i.e., at the layout level, and soft IP, i.e., at the netlist level. ML models require training data to create, which, consequently, also to a large fraction originates in IP. This may lead to two problems. First, IP vendors may claim the (partial) ownership of any model created with their IP, which may not be feasible in practice. Second, and more severely, complex NN models, such as deep NNs, may memorize certain input patterns and allow extraction of parts of the training data. This has been demonstrated in image models that allowed extracting individual training examples [147] (membership inference attack). As an example, we consider a model that is trained to detect lithographic hotspots in layout images. The layout of a memory array is highly regular and, therefore, repeats often in the training data. Additionally, the layout of the memory array may be protected as IP. When the trained lithographic model is released, it may be possible to extract common patterns in the training data, i.e., the protected memory layout. While there are technical solutions to these problems that make extracting training data more difficult, this is mostly a regulatory problem.

The third challenge comes from the portability of models and training. In an ideal world, the ML model trained with data from one tool flow and one technology node would be applicable to designs created in other tool flows and for other technology nodes. In practice, this is less likely to be the case. For instance, Chan *et al.* [148] studied the *noise and chaos* inherent in commercial place&route tools, and importantly also showed that different tools show different susceptibility to small changes in the input. Research on models and training methods that increase retargetability, if successful, would increase the utility and longevity of ML-based CAD tools.

C. Limited Availability of Training Data

A key challenge when employing ML for CAD is the creation of training data. Especially, deep NNs require lots of data. However, not only the amount of data is important but it is crucial that the training data reflect the data observed at inference time. This can only be solved by obtaining training data from a large variety of different designs. Data imbalance, where most of the data belong to one or few classes, exacerbates the problem. This is for instance the case in testing where defects happen rarely. Therefore, available data are often limited, which is circumvented by performing training data augmentation, i.e., create variants of the available data. This may create a false sense of accuracy. A recent case was presented by Reddy *et al.* [149]. They revisited the ICCAD'12 benchmark that is widely used to train and test lithographic hotspot detection [150]. They showed that the high accuracy that was achieved by many state-of-the-art techniques reduces drastically if more examples from a larger variety of designs are introduced. There are several directions to cope with limited training data that are explained in the following.

1) Distributed Learning From Customer Data: Lots of data are created when customers use the CAD tools on their designs. These data may be used to refine the models during usage of the CAD tool via online learning. However, this is only beneficial if data from many customers can be used to train a single model. As outlined earlier, most of these data are subject to IP licenses or confidential, which means it cannot be sent to the developer of the CAD tool for training. Distributed learning (e.g., federated learning [151]) can be a solution, where every customer performs retraining of the tool with its data, and only updated models are exchanged with the tool developer and other customers.

2) Semisupervised Learning and Transfer Learning: Another way to cope with scarcity in training data is to make use of other available data. This could be unlabeled data (semisupervised learning), or data from another but related

problem (transfer learning). Semisupervised learning harnesses unlabeled data to learn the underlying data distribution. First works have successfully followed this approach [91]. Transfer learning exploits the relatedness of problems, where a model is first trained on a related problem (source domain) to serve as a starting point for retraining in the actual problem (target domain). For instance, a model can be trained for one technology and retrained for another [64].

3) Domain Knowledge: CAD looks back on decades of heuristic algorithms that were designed based on domain knowledge of designers. When switching to ML models, this domain knowledge should be harnessed [152]. This is done in parts by deciding which problems to address with ML, but also the design of the models themselves should involve the existing domain knowledge to alleviate the limitedness of training data. Different techniques may be employed to generating the training data itself (e.g., data augmentation), to building the model structure (e.g., restraining NN weights to guarantee monotonicity), and to the postprocessing model outputs (e.g., plausibility checks) [127]. Another option to make use of domain knowledge, is to directly learn from designs that have been implemented by human designers. Some techniques have already been proposed that imitate human designers [52], [65].

D. Interpretability and Adversarial Attacks

ML models, especially NN models, are difficult to debug. While there exist techniques to reverse-engineer NN, this is only possible to some degree. This opens up some challenges.

First, wrong predictions are difficult to explain and also difficult to prevent. This has been demonstrated by Reddy *et al.* [149] as discussed earlier. Second, such models are susceptible to small perturbations in the input. Such perturbations, if selected cleverly, can trick the model to a completely wrong prediction. Liu *et al.* [153] demonstrated adversarial attacks on lithographic hotspot detection. They consider IP vendor that sells fully placed and routed IP. The customer checks the IP on lithographic hotspots using ML classifier. The vendor may for instance aim to trick the classifier to detect no hotspots to make fast profit from low-quality IP. Preventing such attacks can only be done if attacks are already considered during the training to obtain a more robust model. However, more research is still required to achieve (or even guarantee) robustness in models.

Finally, another challenge rises if the model is provided by an untrustworthy source. This is also the case when many users of the CAD tool cooperatively train a model in a distributed setting. A malicious model might work fine at first glance but might have a secret trigger embedded to make it malfunction [154]. An adversary can exploit this to control the output of the model. The IP vendor may train the model to give a seeming advantage to its IP, or may even sabotage the model to work badly on IP from competitors.

VII. Conclusion

This work has given a summary of ML in CAD of ICs. ML promises to fill several gaps in the CAD domain that is still dominated by heuristic algorithms. First, we performed a metastudy of how ML has been used for CAD in the recent five years. We identified several trends, the main ones being a trend toward physical design and manufacturing steps, and a trend toward NN-based models. We presented a categorization of ML in CAD, which is based on how models are used, and discussed state-of-the-art techniques for both design-time and run-time aspects of CAD. Finally, we highlighted the key challenges that need to be solved when employing ML in CAD and outlined directions on how to solve them.

Acknowledgment

The authors thank Victor van Santen and Jannik Prinz for their help in Section IV-G.

References

[1] *International Technology Roadmap for Semiconductors (ITRS) Reports*. Accessed: Jul. 26, 2021. [Online]. Available: http://www.itrs2.net/itrs-reports.html

[2] S. Pagani, P. D. S. Manoj, A. Jantsch, and J. Henkel, "Machine learning for power, energy, and thermal management on multi-core processors: A survey," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 39, no. 1, pp. 101–116, Jan. 2020.

[3] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "ImageNet: A large-scale hierarchical image database," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, 2009, pp. 248–255.

[4] J. Long, E. Shelhamer, and T. Darrell, "Fully convolutional networks for semantic segmentation," in *Proc. Conf. Comput. Vis. Pattern Recognit. (CVPR)*, 2015, pp. 3431–3440.

[5] I. A. M. Elfadel, D. S. Boning, and X. Li, *Machine Learning in VLSI Computer-Aided Design*. Heidelberg, Germany: Springer, 2019.

[6] I. Kononenko and M. Kukar, *Machine Learning and Data Mining*. London, U.K.: Horwood, 2007.

[7] J. Kocijan, R. Murray-Smith, C. E. Rasmussen, and A. Girard, "Gaussian process model based predictive control," in *Proc. IEEE Amer. Control Conf.*, vol. 3, 2004, pp. 2214–2219.

[8] F. Scarselli, M. Gori, A. C. Tsoi, M. Hagenbuchner, and G. Monfardini, "The graph neural network model," *IEEE Trans. Neural Netw.*, vol. 20, no. 1, pp. 61–80, Jan. 2009.

[9] A. Creswell, T. White, V. Dumoulin, K. Arulkumaran, B. Sengupta, and A. A. Bharath, "Generative adversarial networks: An overview," *IEEE Signal Process. Mag.*, vol. 35, no. 1, pp. 53–65, Jan. 2018.

[10] Y.-C. Fang, H.-Y. Lin, M.-Y. Sui, C.-M. Li, and E. J.-W. Fang, "Machine-learning-based dynamic IR drop prediction for ECO," in *Proc. IEEE Int. Conf. Comput.-Aided Design (ICCAD)*, 2018, pp. 1–7.

[11] Y. Zhou *et al.*, "PRIMAL: Power inference using machine learning," in *Proc. Design Autom. Conf. (DAC)*, 2019, p. 39.

[12] C. J. Watkins and P. Dayan, "*Q*-learning," *Mach. Learn.*, vol. 8, nos. 3–4, pp. 279–292, 1992.

[13] M. Z. Alom *et al.*, "The history began from AlexNet: A comprehensive survey on deep learning approaches," 2018, *arXiv:1803.01164*.

[14] V. R. Konda and J. N. Tsitsiklis, "Actor–critic algorithms," in *Proc. Adv. Neural Inf. Process. Syst. (NIPS)*, 2000, pp. 1008–1014.

[15] D. P. Kingma and M. Welling, "Auto-encoding variational Bayes," in *Proc. Int. Conf. Learn. Represent. (ICLR)*, 2013, pp. 1–6.

[16] H. Yang, P. Pathak, F. Gennari, Y.-C. Lai, and B. Yu, "DeePattern: Layout pattern generation with transforming convolutional auto-encoder," in *Proc. ACM Design Autom. Conf. (DAC)*, 2019, p. 148.

[17] G. Mariani, G. Palermo, V. Zaccaria, and C. Silvano, "OSCAR: An optimization methodology exploiting spatial correlation in multicore design spaces," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 31, no. 5, pp. 740–753, May 2012.

[18] B. K. Joardar, R. G. Kim, J. R. Doppa, P. P. Pande, D. Marculescu, and R. Marculescu, "Learning-based application-agnostic 3D NoC design for heterogeneous manycore systems," *IEEE Trans. Comput.*, vol. 68, no. 6, pp. 852–866, Jun. 2019.

[19] A. Deshwal, N. K. Jayakodi, B. K. Joardar, J. R. Doppa, and P. P. Pande, "MOOS: A multi-objective design space exploration and optimization framework for noc enabled manycore systems," *ACM Trans. Embedded Comput. Syst.*, vol. 18, no. 5s, pp. 1–23, 2019.

[20] A. Powell, C. Savvas-Bouganis, and P. Y. Cheung, "High-level power and performance estimation of fpga-based soft processors and its application to design space exploration," *J. Syst. Archit.*, vol. 59, no. 10, pp. 1144–1156, 2013.

[21] H.-Y. Liu and L. P. Carloni, "On learning-based methods for design-space exploration with high-level synthesis," in *Proc. Design Autom. Conf. (DAC)*, 2013, pp. 1–7.

[22] G. Zhong, A. Prakash, S. Wang, Y. Liang, T. Mitra, and S. Niar, "Design space exploration of FPGA-based accelerators with multi-level parallelism," in *Proc. Design Autom. Test Europe Conf. Exhibit. (DATE)*, 2017, pp. 1141–1146.

[23] E. Zennaro, L. Servadei, K. Devarajegowda, and W. Ecker, "A machine learning approach for area prediction of hardware designs from abstract specifications," in *Proc. IEEE Digit. Syst. Design (DSD)*, 2018, pp. 413–420.

[24] S. Dai, Y. Zhou, H. Zhang, E. Ustun, E. F. Young, and Z. Zhang, "Fast and accurate estimation of quality of results in high-level synthesis with machine learning," in *Proc. IEEE Int. Symp. Field Program. Cust. Comput. Mach. (FCCM)*, 2018, pp. 129–132.

[25] E. Ustun, C. Deng, D. Pal, Z. Li, and Z. Zhang, "Accurate operation delay prediction for FPGA HLS using graph neural networks," in *Proc. Int. Conf. Comput.-Aided Design*, 2020, pp. 1–9.

[26] H. Chen and M. Shen, "A deep-reinforcement-learning-based scheduler for FPGA HLS," in *Proc. IEEE Int. Conf. Comput.-Aided Design (ICCAD)*, 2019, pp. 1–8.

[27] Y. Zhang, H. Ren, and B. Khailany, "GRANNITE: Graph neural network inference for transferable power estimation," in *Proc. IEEE Design Autom. Conf. (DAC)*, 2020, pp. 1–6.

[28] G. Pasandi, M. Peterson, M. Herrera, S. Nazarian, and M. Pedram, "Deep-PowerX: A deep learning-based framework for low-power approximate logic synthesis," in *Proc. Int. Symp. Low Power Electron. Design (ISLPED)*, 2020, pp. 73–78.

[29] A. Hosny, S. Hashemi, M. Shalan, and S. Reda, "DRiLLS: Deep reinforcement learning for logic synthesis," in *Proc. IEEE Asia South Pac. Design Autom. Conf. (ASP-DAC)*, 2020, pp. 581–586.

[30] J. Kwon, M. M. Ziegler, and L. P. Carloni, "A learning-based recommender system for autotuning design FIoWs of industrial high-performance processors," in *Proc. IEEE Design Autom. Conf. (DAC)*, 2019, pp. 1–6.

[31] D. Ciregan, U. Meier, and J. Schmidhuber, "Multi-column deep neural networks for image classification," in *Proc. Conf. Comput. Vis. Pattern Recognit. (CVPR)*, 2012, pp. 3642–3649.

[32] J. Fu, H. Zheng, and T. Mei, "Look closer to see better: Recurrent attention convolutional neural network for fine-grained image recognition," in *Proc. Conf. Comput. Vis. Pattern Recognit. (CVPR)*, 2017, pp. 4438–4446.

[33] A. Mirhoseini *et al.*, "Chip placement with deep reinforcement learning," 2020, *arXiv:2004.10746*.

[34] J. Liu, Y. Ding, J. Yang, U. Schlichtmann, and Y. Shi, "Generative adversarial network based scalable on-chip noise sensor placement," in *Proc. IEEE Syst. Chip Conf. (SOCC)*, 2017, pp. 239–242.

[35] E. C. Barboza, N. Shukla, Y. Chen, and J. Hu, "Machine learning-based pre-routing timing prediction with reduced pessimism," in *Proc. ACM Design Autom. Conf. (DAC)*, 2019, pp. 1–6.

[36] Z. Xie *et al.*, "RouteNet: Routability prediction for mixed-size designs using convolutional neural network," in *Proc. IEEE Int. Conf. Comput.-Aided Design (ICCAD)*, 2018, pp. 1–8.

[37] A. F. Tabrizi, N. K. Darav, L. Rakai, I. Bustany, A. Kennings, and L. Behjat, "Eh?Predictor: A deep learning framework to identify detailed routing short violations from a placed netlist," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 39, no. 6, pp. 1177–1190, Jun. 2020.

[38] S. I. Ward *et al.*, "Keep it straight: Teaching placement how to better handle designs with datapaths," in *Proc. Int. Symp. Phys. Design (ISPD)*, 2012, pp. 79–86.

[39] Y. Lin *et al.*, "DREAMPlace: Deep learning toolkit-enabled GPU acceleration for modern VLSI placement," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 40, no. 4, pp. 748–761, Apr. 2021.

[40] J. Lu *et al.*, "ePlace: Electrostatics based placement using Nesterov's method," in *Proc. Design Autom. Conf. (DAC)*, 2014, pp. 1–6.

[41] A. Agnesina, E. Lepercq, J. Escobedo, and S. K. Lim, "Reducing compilation effort in commercial FPGA emulation systems using machine learning," in *Proc. IEEE Int. Conf. Comput.-Aided Design (ICCAD)*, 2019, pp. 1–8.

[42] A. Agnesina, K. Chang, and S. K. Lim, "VLSI placement parameter optimization using deep reinforcement learning," in *Proc. Int. Conf. Comput.-Aided Design (ICCAD)*, 2020, pp. 1–9.

[43] Z. Xie *et al.*, "FIST: A feature-importance sampling and tree-based method for automatic design flow parameter tuning," in *Proc. IEEE Asia South Pac. Design Autom. Conf. (ASP-DAC)*, 2020, pp. 19–25.

[44] Y.-C. Lu, S. S. K. Pentapati, L. Zhu, K. Samadi, and S. K. Lim, "TPGNN: A graph neural network framework for tier partitioning in monolithic 3D ICs," in *Proc. IEEE Design Autom. Conf. (DAC)*, 2020, pp. 1–9.

[45] S. I. Ward *et al.*, "Clock power minimization using structured latch templates and decision tree induction," in *Proc. IEEE Int. Conf. Comput.-Aided Design (ICCAD)*, 2013, pp. 599–606.

[46] Y.-C. Lu, J. Lee, A. Agnesina, K. Samadi, and S. K. Lim, "GAN-CTS: A generative adversarial framework for clock tree prediction and optimization," in *Proc. IEEE Int. Conf. Comput.-Aided Design (ICCAD)*, 2019, pp. 1–8.

[47] C. Yu and Z. Zhang, "Painting on placement: Forecasting routing congestion using conditional generative adversarial nets," in *Proc. Design Autom. Conf. (DAC)*, 2019, p. 219.

[48] M. B. Alawieh, W. Li, Y. Lin, L. Singhal, M. A. Iyer, and D. Z. Pan, "High-definition routing congestion prediction for large-scale FPGAs," in *Proc. IEEE Asia South Pac. Design Autom. Conf. (ASP-DAC)*, 2020, pp. 26–31.

[49] T.-C. Yu *et al.*, "Pin accessibility prediction and optimization with deep learning-based pin pattern recognition," in *Proc. Design Autom. Conf. (DAC)*, 2019, pp. 1–6.

[50] W.-T. Hung, J.-Y. Huang, Y.-C. Chou, C.-H. Tsai, and M. Chao, "Transforming global routing report into DRC violation map with convolutional neural network," in *Proc. Int. Symp. Phys. Design (ISPD)*, 2020, pp. 57–64.

[51] R. Liang *et al.*, "DRC hotspot prediction at sub-10nm process nodes using customized convolutional network," in *Proc. Int. Symp. Phys. Design (ISPD)*, 2020, pp. 135–142.

[52] K. Zhu *et al.*, "GeniusRoute: A new analog routing paradigm using generative neural network guidance," in *Proc. IEEE Int. Conf. Comput.-Aided Design (ICCAD)*, 2019, pp. 1–8.

[53] T. Qu, Y. Lin, Z. Lu, Y. Su, and Y. Wei, "Asynchronous reinforcement learning framework for net order exploration in detailed routing," in *Proc. Design Autom. Test Eurpoe (DATE)*, Feb. 2021, pp. 1815–1820.

[54] H. Li, G. Chen, B. Jiang, J. Chen, and E. F. Young, "Dr.CU 2.0: A scalable detailed routing framework with correct-by-construction design rule satisfaction," in *Proc. IEEE Int. Conf. Comput.-Aided Design (ICCAD)*, 2019, pp. 1–7.

[55] H. Li *et al.*, "Attacking split manufacturing from a deep learning perspective," in *Proc. ACM Design Autom. Conf. (DAC)*, 2019, pp. 1–6.

[56] W. Zeng, B. Zhang, and A. Davoodi, "Analysis of security of split manufacturing using machine learning," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 27, no. 12, pp. 2767–2780, Dec. 2019.

[57] C. Ho and A. B. Kahng, "IncPIRD: Fast learning-based prediction of incremental IR drop," in *Proc. Int. Conf. Comput.-Aided Design (ICCAD)*, 2019, pp. 1–8.

[58] Z. Xie *et al.*, "PowerNet: Transferable dynamic IR drop estimation via maximum convolutional neural network," in *Proc. Asia South Pac. Design Autom. Conf. (ASP-DAC)*, 2020, pp. 13–18.

[59] H. Zhou, W. Jin, and S. X. Tan, "GridNet: Fast data-driven EMInduced IR drop prediction and localized fixing for on-chip power grid networks," in *Proc. Int. Conf. Comput.-Aided Design (ICCAD)*, 2020, pp. 1–9.

[60] V. A. Chhabria, A. B. Kahng, M. Kim, U. Mallappa, S. S. Sapatnekar, and B. Xu, "Template-based PDN synthesis in floorplan and placement using classifier and CNN techniques," in *Proc. Asia South Pac. Design Autom. Conf. (ASP-DAC)*, 2020, pp. 44–49.

[61] Y. Cao *et al.*, "Learning-based prediction of package power delivery network quality," in *Proc. ACM Asia South Pac. Design Autom. Conf. (ASP-DAC)*, 2019, pp. 160–166.

[62] T. Dhar *et al.*, "ALIGN: A system for automating analog layout," *IEEE Des. Test*, vol. 38, no. 2, pp. 8–18, Apr. 2021.

[63] H. Chen *et al.*, "MAGICAL: An open-source fully automated analog ic layout system from netlist to GDSII," *IEEE Des. Test*, vol. 38, no. 2, pp. 19–26, Apr. 2021.

[64] H. Wang *et al.*, "GCN-RL circuit designer: Transferable transistor sizing with graph neural networks and reinforcement learning," in *Proc. Design Autom. Conf. (DAC)*, 2020, pp. 1–6.

[65] B. Xu *et al.*, "WellGAN: Generative-adversarial-network-guided well generation for analog/mixed-signal circuit layout," in *Proc. ACM Design Autom. Conf. (DAC)*, 2019, pp. 1–6.

[66] Y. Li *et al.*, "A customized graph neural network model for guiding analog IC placement," in *Proc. IEEE Int. Conf. Comput.-Aided Design (ICCAD)*, 2020, pp. 1–9.

[67] B. Shook, P. Bhansali, C. Kashyap, C. Amin, and S. Joshi, "MLParest: Machine learning based parasitic estimation for custom circuit design," in *Proc. IEEE Design Autom. Conf. (DAC)*, 2020, pp. 1–6.

[68] H. Ren, G. F. Kokai, W. J. Turner, and T.-S. Ku, "ParaGraph: Layout parasitics and device parameter prediction using graph neural networks," in *Proc. IEEE Design Autom. Conf. (DAC)*, 2020, pp. 1–6.

[69] K. Hakhamaneshi, N. Werblun, P. Abbeel, and V. Stojanović, "BagNet: Berkeley analog generator with layout optimizer boosted with deep neural networks," in *Proc. IEEE Int. Conf. Comput.-Aided Design (ICCAD)*, 2019, pp. 1–8.

[70] M. B. Alawieh, Y. Lin, Z. Zhang, M. Li, Q. Huang, and D. Z. Pan, "GAN-SRAF: Sub-resolution assist feature generation using generative adversarial networks," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 40, no. 2, pp. 373–385, Feb. 2021.

[71] X. Xu *et al.*, "Subresolution assist feature generation with supervised data learning," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 37, no. 6, pp. 1225–1236, Jun. 2018.

[72] *Calibre Verification User's Manual*, Mentor Graph., Wilsonville, OR, USA, 2008.

[73] H. Yang, S. Li, Z. Deng, Y. Ma, B. Yu, and E. F. Young, "GAN-OPC: Mask optimization with lithography-guided generative adversarial nets," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 39, no. 10, pp. 2822–2834, Oct. 2020.

[74] J.-R. Gao, X. Xu, B. Yu, and D. Z. Pan, "MOSAIC: Mask optimizing solution with process window aware inverse correction," in *Proc. IEEE Design Autom. Conf. (DAC)*, 2014, pp. 1–6.

[75] B. Jiang, L. Liu, Y. Ma, H. Zhang, E. F. Young, and B. Yu, "Neural-ILT: Migrating ILT to nerual networks for mask printability and complexity co-optimizaton," in *Proc. Int. Conf. Comput.-Aided Design (ICCAD)*, Nov. 2020, pp. 1–9.

[76] G. Chen, W. Chen, Y. Ma, H. Yang, and B. Yu, "DAMO: Deep agile mask optimization for full chip scale," in *Proc. Int. Conf. Comput.-Aided Design (ICCAD)*, Nov. 2020, pp. 1–9.

[77] Y. Watanabe, T. Kimura, T. Matsunawa, and S. Nojima, "Accurate lithography simulation model based on convolutional neural networks," in *Proc. Soc. Photo-Opt. Instrum. Eng. (SPIE) Conf.*, 2017, Art. no. 10454.

[78] *Synopsys Sentaurus Lithography*. [Online]. Available: https://www.synopsys.com/silicon/mask-synthesis/sentauruslithography.html (Accessed: Jul. 26, 2021).

[79] W. Ye, M. B. Alawieh, Y. Lin, and D. Z. Pan, "LithoGAN: End-to-end lithography modeling with generative adversarial networks," in *Proc. ACM Design Autom. Conf. (DAC)*, 2019, pp. 1–6.

[80] W. Ye, M. B. Alawieh, Y. Watanabe, S. Nojima, Y. Lin, and D. Z. Pan, "TEMPO: Fast mask topography effect modeling with deep learning," in *Proc. Int. Symp. Phys. Design (ISPD)*, Taipei, Taiwan, Sep. 2020, pp. 127–134.

[81] Y. Lin *et al.*, "Data efficient lithography modeling with transfer learning and active data selection," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 38, no. 10, pp. 1900–1913, Oct. 2019.

[82] M. Shin and J.-H. Lee, "Accurate lithography hotspot detection using deep convolutional neural networks," *J. Micro/Nanolithography*, vol. 15, no. 4, 2016, Art. no. 043507.

[83] H. Yang, L. Luo, J. Su, C. Lin, and B. Yu, "Imbalance aware lithography hotspot detection: A deep learning approach," *J. Micro/Nanolithography*, vol. 16, no. 3, 2017, Art. no. 1014807.

[84] H. Yang, J. Su, Y. Zou, Y. Ma, B. Yu, and E. F. Y. Young, "Layout hotspot detection with feature tensor generation and deep biased learning," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 38, no. 6, pp. 1175–1187, Jun. 2019.

[85] H. Yang, Y. Lin, B. Yu, and E. F. Young, "Lithography hotspot detection: From shallow to deep learning," in *Proc. IEEE Syst. Chip Conf. (SOCC)*, 2017, pp. 233–238.

[86] J. Chen *et al.*, "Lithography hotspot detection using a double inception module architecture," *J. Micro/Nanolithography*, vol. 18, no. 1, 2019, Art. no. 013507.

[87] Y. Jiang, F. Yang, B. Yu, D. Zhou, and X. Zeng, "Efficient layout hotspot detection via binarized residual neural network ensemble," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 40, no. 7, pp. 1476–1488, Jul. 2021.

[88] H. Zhang, B. Yu, and E. F. Young, "Enabling online learning in lithography hotspot detection with information-theoretic feature optimization," in *Proc. Int. Conf. Comput.-Aided Design (ICCAD)*, 2016, pp. 1–8.

[89] W. Ye, M. B. Alawieh, M. Li, Y. Lin, and D. Z. Pan, "Litho-GPA: Gaussian process assurance for lithography hotspot detection," in *Proc. Design Autom. Test Europe Conf. Exhibit. (DATE)*, Florence, Italy, Mar. 2019, pp. 54–59.

[90] H. Yang, S. Li, C. Tabery, B. Lin, and B. Yu, "Bridging the gap between layout pattern sampling and hotspot detection via batch active learning," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 40, no. 7, pp. 1464–1475, Jul. 2021.

[91] Y. Chen, Y. Lin, T. Gai, Y. Su, Y. Wei, and D. Z. Pan, "Semi-supervised hotspot detection with self-paced multi-task learning," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 39, no. 7, pp. 1511–1523, Jul. 2020.

[92] R. Chen *et al.*, "Faster region-based hotspot detection," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, early access, Sep. 3, 2020, doi: 10.1109/TCAD.2020.3021663.

[93] A. Ciccazzo, G. Di Pillo, and V. Latorre, "A SVM surrogate model-based method for parametric yield optimization," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 35, no. 7, pp. 1224–1228, Jul. 2016.

[94] K. Nakata, R. Orihara, Y. Mizuoka, and K. Takagi, "A comprehensive big-data-based monitoring system for yield enhancement in semiconductor manufacturing," *IEEE Trans. Semicond. Manuf.*, vol. 30, no. 4, pp. 339–344, Nov. 2017.

[95] M. B. Alawieh, F. Wang, and X. Li, "Identifying wafer-level systematic failure patterns via unsupervised learning," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 37, no. 4, pp. 832–844, Apr. 2018.

[96] Z. Gao, J. Tao, Y. Su, D. Zhou, X. Zeng, and X. Li, "Efficient rare failure analysis over multiple corners via correlated Bayesian inference," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 39, no. 10, pp. 2029–2041, Oct. 2020.

[97] M. B. Alawieh, D. Boning, and D. Z. Pan, "Wafer map defect patterns classification using deep selective learning," in *Proc. IEEE Design Autom. Conf. (DAC)*, 2020, pp. 1–6.

[98] W. Chen, S. Ray, J. Bhadra, M. Abadir, and L.-C. Wang, "Challenges and trends in modern SoC design verification," *IEEE Des. Test*, vol. 34, no. 5, pp. 7–22, Oct. 2017.

[99] S. Li and B. Jacob, "Statistical DRAM modeling," in *Proc. Int. Symp. Memory Syst. (MEMSYS)*, 2019, pp. 521–530.

[100] D. Lee and A. Gerstlauer, "Learning-based, fine-grain power modeling of system-level hardware IPs," *ACM Trans. Design Autom. Electron. Syst.*, vol. 23, no. 3, pp. 1–25, 2018.

[101] W. Chen, K.-K. Hsieh, L.-C. Wang, and J. Bhadra, "Data-driven test plan augmentation for platform verification," *IEEE Des. Test*, vol. 34, no. 5, pp. 23–29, Oct. 2017.

[102] Y. Ma *et al.*, "High performance graph convolutional networks with applications in testability analysis," in *Proc. Annu. Design Autom. Conf.*, 2019, pp. 1–18.

[103] D. Kim, P. Kang, S. Cho, H.-J. Lee, and S. Doh, "Machine learning-based novelty detection for faulty wafer detection in semiconductor manufacturing," *Exp. Syst. Appl.*, vol. 39, no. 4, pp. 4075–4083, 2012.

[104] A. DeOrio, Q. Li, M. Burgess, and V. Bertacco, "Machine learningbased anomaly detection for post-silicon bug diagnosis," in *Proc. Design Autom. Test Europe (DATE)*, 2013, pp. 491–496.

[105] H. Hu, P. Li, and J. Z. Huang, "Parallelizable Bayesian optimization for analog and mixed-signal rare failure detection with high coverage," in *Proc. IEEE Int. Conf. Comput.-Aided Design (ICCAD)*, 2018, pp. 1–8.

[106] P. B. L. Meijer, *Neural Network Applications in Device and Subcircuit Modelling for Circuit Simulation*, Philips Electron., Amsterdam, The Netherlands, 1996.

[107] Z. Zhang *et al.*, "New-generation design-technology co-optimization (DTCO): Machine-learning assisted modeling framework," in *2019 Silicon Nanoelectronics Workshop (SNW 2019)*. Red Hook, NY, USA: Curran Assoc., Inc., Jun. 2019, pp. 1–2.

[108] K. Lamamra and S. Berrah, "Modeling of MOSFET transistor by MLP neural networks," in *Recent Advances in Electrical Engineering and Control Applications*, M. Chadli, S. Bououden, and I. Zelinka, Eds. Cham, Switzerland: Springer Int., 2017, pp. 407–415.

[109] L. Zhang and M. Chan, "Artificial neural network design for compact modeling of generic transistors," *J. Comput. Electron.*, vol. 16, no. 3, pp. 825–832, Sep. 2017.

[110] M. Li, O. ßIrsoy, C. Cardie, and H. G. Xing, "Physics-inspired neural networks for efficient device compact modeling," *IEEE J. Explor. Solid-State Computat. Devices Circuits*, vol. 2, pp. 44–49, 2016.

[111] F. Klemme, J. Prinz, V. M. van Santen, J. Henkel, and H. Amrouch, "Modeling emerging technologies using machine learning: Challenges and opportunities," in *Proc. Int. Conf. Comput.-Aided Design (ICCAD)*, 2020, pp. 1–9.

[112] F. Klemme, Y. Chauhan, J. Henkel, and H. Amrouch, "Cell library characterization using machine learning for design technology co-optimization," in *Proc. IEEE Int. Conf. Comput.-Aided Design (ICCAD)*, 2020, pp. 1–9.

[113] T. Ebi, D. Kramer, W. Karl, and J. Henkel, "Economic learning for thermal-aware power budgeting in many-core architectures," in *Proc. ACM Conf. Hardw. Softw. Codesign Syst. Synth. (CODES)*, 2011, pp. 189–196.

[114] H. Shen, Y. Tan, J. Lu, Q. Wu, and Q. Qiu, "Achieving autonomous power management using reinforcement learning," *ACM Trans. Design Autom. Electron. Syst.*, vol. 18, no. 2, pp. 1–32, 2013.

[115] R. A. Shafik, S. Yang, A. Das, L. A. Maeda-Nunez, G. V. Merrett, and B. M. Al-Hashimi, "Learning transfer-based adaptive energy minimization in embedded systems," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 35, no. 6, pp. 877–890, Jun. 2016.

[116] T. Kim, Z. Sun, H.-B. Chen, H. Wang, and S. X.-D. Tan, "Energy and lifetime optimizations for dark silicon manycore microprocessor considering both hard and soft errors," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 25, no. 9, pp. 2561–2574, Sep. 2017.

[117] S. M. P. Dinakarrao, A. Joseph, A. Haridass, M. Shafique, J. Henkel, and H. Homayoun, "Application and thermal-reliability-aware reinforcement learning based multi-core power management," *J. Emerg. Technol. Comput. Syst.*, vol. 15, no. 4, pp. 1–19, 2019.

[118] U. Gupta, S. K. Mandal, M. Mao, C. Chakrabarti, and U. Y. Ogras, "A deep *Q*-learning approach for dynamic management of heterogeneous processors," *Comput. Architect. Lett.*, vol. 18, no. 1, pp. 14–17, 2019.

[119] Z. Chen and D. Marculescu, "Distributed reinforcement learning for power limited many-core system performance optimization," in *Proc. Design Autom. Test Europe Conf. Exhibit. (DATE)*, 2015, pp. 1521–1526.

[120] H. Li, Z. Tian, R. K. Maeda, X. Chen, J. Feng, and J. Xu, "Co-manage power delivery and consumption for manycore systems using reinforcement learning," in *Proc. ACM Int. Conf. Comput.-Aided Design (ICCAD)*, 2018, pp. 1–8.

[121] H. Mao, M. Alizadeh, I. Menache, and S. Kandula, "Resource management with deep reinforcement learning," in *Proc. ACM Workshop Hot Topics Netw. (HotNets)*, 2016, pp. 50–56.

[122] F. M. M. ul Islam and M. Lin, "Hybrid DVFS scheduling for real-time systems based on reinforcement learning," *Syst. J.*, vol. 11, no. 2, pp. 931–940, 2015.

[123] J.-G. Park, N. Dutt, and S.-S. Lim, "ML-Gov: A machine learning enhanced integrated CPU-GPU DVFS governor for mobile gaming," in *Proc. ACM Symp. Embedded Syst. Real Time Multimedia (ESTImedia)*, 2017, pp. 12–21.

[124] S. K. Mandal, G. Bhat, C. A. Patil, J. R. Doppa, P. P. Pande, and U. Y. Ogras, "Dynamic resource management of heterogeneous mobile platforms via imitation learning," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 27, no. 12, pp. 2842–2854, Dec. 2019.

[125] R. G. Kim *et al.*, "Imitation learning for dynamic VFI control in large-scale manycore systems," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 25, no. 9, pp. 2458–2471, Sep. 2017.

[126] S. Ross, G. Gordon, and D. Bagnell, "A reduction of imitation learning and structured prediction to no-regret online learning," in *Proc. Int. Conf. Artif. Intell. Stat. (AISTATS)*, 2011, pp. 627–635.

[127] M. Rapp, H. Amrouch, M. C. Wolf, and J. Henkel, "Machine learning techniques to support many-core resource management: Challenges and opportunities," in *Proc. ACM/IEEE Workshop Mach. Learn. CAD (MLCAD)*, 2019, pp. 1–9.

[128] D. Amodei, C. Olah, J. Steinhardt, P. Christiano, J. Schulman, and D. Mané, "Concrete problems in AI safety," 2016, *arXiv:1606.06565*.

[129] W. L. Bircher, M. Valluri, J. Law, and L. K. John, "Runtime identification of microprocessor energy saving opportunities," in *Proc. IEEE Int. Symp. Low Power Electron. Design (ISLPED)*, 2005, pp. 275–280.

[130] M. Sagi, N. A. V. Doan, M. Rapp, T. Wild, J. Henkel, and A. Herkersdorf, "A lightweight nonlinear methodology to accurately model multicore processor power," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 39, no. 11, pp. 3152–3164, Nov. 2020.

[131] S. Sadiqbatcha, H. Zhao, H. Amrouch, J. Henkel, and S. X.-D. Tan, "Hot spot identification and system parameterized thermal modeling for multi-core processors through infrared thermal imaging," in *Proc. IEEE Design Autom. Test Europe (DATE)*, 2019, pp. 48–53.

[132] D. Kim, J. Zhao, J. Bachrach, and K. Asanović, "SIMMANI: Runtime power modeling for arbitrary RTL with automatic signal selection," in *Proc. Int. Symp. Microarchit. (MICRO)*, 2019, pp. 1050–1062.

[133] M. Rapp, M. B. Sikal, H. Khdr, and J. Henkel, "SmartBoost: Lightweight ML-driven boosting for thermally-constrained many-core processors," in *Proc. Design Autom. Conf. (DAC)*, 2021, pp. 1–9.

[134] M. Rapp, A. Pathania, T. Mitra, and J. Henkel, "Neural network-based performance prediction for task migration on S-NUCA many-cores," *IEEE Trans. Comput.*, vol. 70, no. 10, pp. 1691–1704, Oct. 2021.

[135] U. Gupta, M. Babu, R. Ayoub, M. Kishinevsky, F. Paterna, and U. Y. Ogras, "STAFF: Online learning with stabilized adaptive forgetting factor and feature selection algorithm," in *Proc. IEEE Design Autom. Conf. (DAC)*, 2018, pp. 1–6.

[136] Y. Kim, P. Mercati, A. More, E. Shriver, and T. Rosing, "P4: Phase-based power/performance prediction of heterogeneous systems via neural networks," in *Proc. IEEE Int. Conf. Comput.-Aided Design (ICCAD)*, 2017, pp. 683–690.

[137] K. Zhang *et al.*, "Machine learning-based temperature prediction for runtime thermal management across system components," *IEEE Trans. Parallel Distrib. Syst.*, vol. 29, no. 2, pp. 405–419, Feb. 2018.

[138] J. M. N. Abad and A. Soleimani, "Novel feature selection algorithm for thermal prediction model," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 26, no. 10, pp. 1831–1844, Oct. 2018.

[139] M. Sagi *et al.*, "Long short-term memory neural network-based power forecasting of multi-core processors," in *Proc. Design Autom. Test Europe Conf. Exhibit. (DATE)*, 2021, pp. 1685–1690.

[140] L. Li *et al.*, "MADS: A framework for design and implementation of adaptive digital predistortion systems," *IEEE J. Emerg. Sel. Topics Circuits Syst.*, vol. 9, no. 4, pp. 712–722, Dec. 2019.

[141] A. Jonsson and A. Barto, "Causal graph based decomposition of factored MDPs," *J. Mach. Learn. Res.*, vol. 7, pp. 2259–2301, Dec. 2006.

[142] A. Bhuiyan, F. Reghenzani, W. Fornaciari, and Z. Guo, "Optimizing energy in non-preemptive mixed-criticality scheduling by exploiting probabilistic information," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 39, no. 11, pp. 3906–3917, Nov. 2020.

[143] Y. Bengio, A. Lodi, and A. Prouvost, "Machine learning for combinatorial optimization: A methodological tour D'Horizon," *Eur. J. Oper. Res.*, vol. 290, no. 2, pp. 405–421, 2021.

[144] O. Vinyals, M. Fortunato, and N. Jaitly, "Pointer networks," in *Proc. Neural Inf. Process. Syst. (NeurIPS)*, 2015, pp. 2692–2700.

[145] A. Vaswani *et al.*, "Attention is all you need," in *Proc. Neural Inf. Process. Syst. (NeurIPS)*, 2017, pp. 5998–6008.

[146] P. Emami and S. Ranka, "Learning permutations with Sinkhorn policy gradient," 2018, *arXiv:1805.07010*.

[147] J. Hayes, L. Melis, G. Danezis, and E. De Cristofaro, "LOGAN: Membership inference attacks against generative models," *Proc. Privacy Enhanc. Technol.*, vol. 2019, no. 1, pp. 133–152, 2019.

[148] T.-B. Chan, A. B. Kahng, and M. Woo, "Revisiting inherent noise floors for interconnect prediction," in *Proc. Workshop Syst. Level Interconnect Probl. Pathfinding Workshop (SLIP)*, 2020, pp. 1–7.

[149] G. R. Reddy, K. Madkour, and Y. Makris, "Machine learning-based hotspot detection: Fallacies, pitfalls and marching orders," in *Proc. IEEE Int. Conf. Comput.-Aided Design (ICCAD)*, 2019, pp. 1–8.

[150] J. A. Torres, "ICCAD-2012 CAD contest in fuzzy pattern matching for physical verification and benchmark suite," in *Proc. IEEE Int. Conf. Comput.-Aided Design (ICCAD)*, 2012, pp. 349–350.

[151] H. B. McMahan *et al.*, "Communication-efficient learning of deep networks from decentralized data," in *Proc. Int. Conf. Artif. Intell. Stat. (AISTATS)*, 2017, pp. 1273–1282.

[152] J. R. Doppa, J. Rosca, and P. Bogdan, "Autonomous design space exploration of computing systems for sustainability: Opportunities and challenges," *IEEE Des. Test*, vol. 36, no. 5, pp. 35–43, Oct. 2019.

[153] K. Liu *et al.*, "Are adversarial perturbations a showstopper for ML-based CAD? A case study on CNN-based lithographic hotspot detection," 2019, *arXiv:1906.10773*.

[154] T. Gu, K. Liu, B. Dolan-Gavitt, and S. Garg, "BadNets: Evaluating backdooring attacks on deep neural networks," *IEEE Access*, vol. 7, pp. 47230–47244, 2019.

Martin Rapp (Graduate Student Member, IEEE) received the B.Sc. and M.Sc. degrees (with Distinction) in computer science from the Karlsruhe Institute of Technology, Karlsruhe, Germany, in 2014 and 2016, respectively, where he is currently pursuing the Ph.D. degree under the supervision of Dr. J. Henkel.

His current research focuses on resource-constrained machine learning: ML-based runtime resource management for many-core architectures and distributed resource-aware on-device training of neural networks.

Hussam Amrouch (Member, IEEE) received the Ph.D. degree (with Distinction, *summa cum laude*) from the Karlsruhe Institute of Technology (KIT), Karlsruhe, Germany, in 2015.

He is a Junior Professor heading the Chair of Semiconductor Test and Reliability, Computer Science, Electrical Engineering Faculty, University of Stuttgart, Stuttgart, Germany, as well as a Research Group Leader with KIT. He has over 140 publications (including 55 journals) in multidisciplinary research areas across the entire computing stack, starting from semiconductor physics to circuit design all the way up to computer-aided design and computer architecture. His main research interests are design for reliability and testing from device physics to systems, machine learning, security, approximate computing, and emerging technologies with a special focus on ferroelectric devices.

Dr. Amrouch holds eight HiPEAC Paper Awards and four best paper nominations at top EDA conferences: DAC'16, DAC'17, DATE'17, and EDTM'21 for his work on reliability. He currently serves as an Associate Editor for *Integration, the VLSI Journal*. He has served in the technical program committees of many major EDA conferences, such as DAC, ASP-DAC, and ICCAD, and as a Reviewer in many top journals, such as IEEE TRANSACTIONS ON ELECTRON DEVICES, IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS—PART I: REGULAR PAPERS, IEEE TRANSACTIONS ON VERY LARGE SCALE INTEGRATION (VLSI) SYSTEMS, IEEE TRANSACTIONS ON COMPUTER-AIDED DESIGN OF INTEGRATED CIRCUITS AND SYSTEMS, and IEEE TRANSACTIONS ON COMPUTERS.

Yibo Lin (Member, IEEE) received the B.S. degree in microelectronics from Shanghai Jiaotong University, Shanghai, China, in 2013, and the Ph.D. degree from the Electrical and Computer Engineering Department, University of Texas at Austin, Austin, TX, USA, in 2018.

He is currently an Assistant Professor with the Computer Science Department associated with the Center for Energy-Efficient Computing and Applications, Peking University, Beijing, China. His research interests include physical design, machine learning applications, GPU acceleration, and hardware security.

Dr. Lin has received five Best Paper Awards at premier venues (TCAD 2021, ISPD 2020, DAC 2019, VLSI Integration 2018, and SPIE 2016). He has also served in the Technical Program Committees of many major conferences, including ICCAD, ICCD, ISPD, and DAC.

Bei Yu (Member, IEEE) received the Ph.D. degree from The University of Texas at Austin, Austin, TX, USA, in 2014.

He is currently an Associate Professor with the Department of Computer Science and Engineering, The Chinese University of Hong Kong, Hong Kong.

Dr. Yu received seven Best Paper Awards from ASPDAC 2021, ICTAI 2019, *Integration, the VLSI Journal* in 2018, ISPD 2017, SPIE Advanced Lithography Conference 2016, ICCAD 2013, and ASPDAC 2012, and six ICCAD/ISPD Contest Awards. He has served as the TPC Chair of ACM/IEEE Workshop on Machine Learning for CAD, and in many journal editorial boards and conference committees. He is an Editor of IEEE TCCPS Newsletter.

David Z. Pan (Fellow, IEEE) received the B.S. degree from Peking University, Beijing, China, in 1992, and the M.S. and Ph.D. degrees from the University of California at Los Angeles, Los Angeles, CA, USA, in 1994, 1998, and 2000, respectively.

From 2000 to 2003, he was a Research Staff Member with IBM T. J. Watson Research Center, Yorktown Heights, NY, USA. He is currently the Silicon Labs Endowed Chair Professor with the Department of Electrical and Computer Engineering, The University of Texas at Austin, Austin, TX, USA. He has published over 420 journal articles and refereed conference papers, and is the holder of eight U.S. patents. His research interests include bidirectional AI and IC interactions, electronic design automation, design for manufacturing, and CAD for analog/mixed-signal ICs and emerging technologies.

Dr. Pan has received a number of awards, including the SRC Technical Excellence Award in 2013, the DAC Top 10 Author in Fifth Decade, the DAC Prolific Author Award, the ASP-DAC Frequently Cited Author Award, 20 Best Paper Awards (TCAD 2021, ISPD 2020, ASPDAC 2020, DAC 2019, GLSVLSI 2018, VLSI Integration 2018, HOST 2017, SPIE 2016, ISPD 2014, ICCAD 2013, ASPDAC 2012, ISPD 2011, IBM Research 2010 Pat Goldberg Memorial Best Paper Award, ASPDAC 2010, DATE 2009, ICICDT 2009, and SRC Techcon in 1998, 2007, 2012, and 2015) and 18 additional Best Paper Award nominations, Communications of the ACM Research Highlights in 2014, the UT Austin RAISE Faculty Excellence Award in 2014, the Cadence Academic Collaboration Award in 2019, and many international CAD contest awards, among others. He has served in many journal editorial boards and conference committees, including various leadership roles, such as the ICCAD 2019 General Chair, ASP-DAC 2017 TPC Chair, and ISPD 2008 General Chair. He is a Fellow of SPIE.

Marylin Wolf (Fellow, IEEE) received the B.S., M.S., and Ph.D. degrees in electrical engineering from Stanford University, Stanford, CA, USA, in 1980, 1981, and 1984, respectively.

She is Elmer E. Koch Professor of Engineering and Chair of the Department of Computer Science and Engineering, University of Nebraska–Lincoln, Lincoln, NE, USA. She was with AT&T Bell Laboratories, Atlanta, GA, USA, from 1984 to 1989. She was with the Faculty of Princeton University, Princeton, NJ, USA, from 1989 to 2007 and was a Farmer Distinguished Chair with Georgia Tech, Atlanta, from 2007 to 2019. Her research interests include cyber–physical systems, embedded computing, embedded video and computer vision, and VLSI systems.

Dr. Wolf has received the IEEE Computer Society Goode Memorial Award, the ASEE Terman Award, and the IEEE Circuits and Systems Society Education Award. She is a Fellow of ACM and an IEEE Computer Society Golden Core Member.

Jörg Henkel (Fellow, IEEE) received the Diploma and Ph.D. degrees *(summa cum laude)* from the Technical University of Braunschweig, Braunschweig, Germany.

He was a Research Staff Member with NEC Laboratories, Princeton, NJ, USA. He is the Chair Professor of Embedded Systems with the Karlsruhe Institute of Technology, Karlsruhe, Germany. His research work is focused on co-design for embedded hardware/software systems with respect to power, thermal, and reliability aspects.

Dr. Henkel has received six Best Paper Awards throughout his career from, among others, ICCAD, ESWeek, and DATE. For two consecutive terms, he served as the Editor-in-Chief for the *ACM Transactions on Embedded Computing Systems*. He is currently the Editor-in-Chief of the IEEE DESIGN&TEST. He is/has been an Associate Editor for major ACM and IEEE journals. He has led several conferences as a General Chair, including ICCAD and ESWeek, and serves as a Steering Committee Chair/Member for leading conferences and journals for embedded and cyber–physical systems. He coordinates the DFG Program SPP 1500 "Dependable Embedded Systems" and is a Site Coordinator of the DFG TR89 collaborative research center on "Invasive Computing." He is the Chairman of the IEEE Computer Society, Germany Chapter.