Special Session: Machine Learning for Embedded System Design

Erika S. Alcorta
Andreas Gerstlauer
The University of Texas at Austin
Austin, TX, USA
{esalcort,gerstl}@utexas.edu

Chenhui Deng
Qi Sun
Zhiru Zhang
Cornell University
Ithaca, NY, USA
{cd574,qs228,zhiruz}@cornell.edu

Ceyu Xu
Lisa Wu Wills
Duke University
Durham, NC, USA
ceyu.xu@duke.edu,lisa@cs.duke.edu

Daniela Sánchez Lopera
Wolfgang Ecker
Infineon Technologies AG
Technical University of Munich
Munich, Germany
{firstname.lastnames}@infineon.com

Siddharth Garg
New York University
New York, NY, USA
siddharth.j.garg@gmail.com

Jiang Hu
Texas A&M University
College Station, TX, USA
jianghu@tamu.edu

ABSTRACT

Embedded systems are becoming increasingly complex, which has led to a productivity crisis in their design and verification. Although conventional design automation coupled with IP and platform reuse techniques have led to leaps in design productivity improvement, they face fundamental limits given that most design optimization and verification problems remain NP-hard and that reuse of predesigned IP blocks and platforms inherently limits flexibility and optimality. At the same time, machine learning (ML) has recently made unprecedented advances and created phenomenal impact in various computing applications. In particular, application of ML techniques as a way to extract knowledge and learn from existing design, optimization and verification data has recently seen a lot of excitement and promise at lower physical and integrated circuit levels of abstraction. Using ML has the potential to similarly close the complexity gap in embedded system design, but corresponding ML-based approaches for embedded system optimization and verification at higher levels of abstraction are still at their infancy.

This paper presents the current state of the art, along with opportunities and open challenges, in the application of ML methods for embedded system design and optimization. We discuss design and optimization at different levels of abstraction ranging from system-level modeling and optimization and high-level synthesis to RTL and micro-architecture design, bringing together perspectives from different communities in both academia and industry.

CCS CONCEPTS

• **Computer systems organization → Embedded systems**; • **Hardware → Methodologies for EDA**.

CODES/ISSS '23 Companion, September 17–22, 2023, Hamburg, Germany

ACM ISBN 979-8-4007-0289-1/23/09.
https://doi.org/10.1145/3607888.3608962

KEYWORDS

Machine Learning, Embedded System Design

ACM Reference Format:
Erika S. Alcorta, Andreas Gerstlauer, Chenhui Deng, Qi Sun, Zhiru Zhang, Ceyu Xu, Lisa Wu Wills, Daniela Sánchez Lopera, Wolfgang Ecker, Siddharth Garg, and Jiang Hu. 2023. Special Session: Machine Learning for Embedded System Design. In *2023 International Conference on Hardware/Software Codesign and System Synthesis (CODES/ISSS '23 Companion), September 17–22, 2023, Hamburg, Germany*. ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3607888.3608962

1 INTRODUCTION

With the increasing complexity and heterogeneity of processors and applications, embedded systems design and runtime management have become more challenging than ever before. The vast design decision space makes it impractical to exhaustively explore every possibility. Thus, intelligent and fast solution search has been essential to finding highly optimized designs.

At any level of abstraction in the design process, two main components are required: a decision-maker or optimizer, and a predictive model or cost function, as shown in Figure 1. The optimization can be performed through either constructive or iterative search. A constructive algorithm starts with a blank solution and makes one decision at a time until a complete solution is obtained, e.g., using greedy heuristics. An iterative approach starts from a complete candidate design that is gradually refined to improve cost metrics, e.g., using meta-heuristics such as simulated annealing. In either case, the predictive model evaluates a partial or complete design candidate and outputs estimations of the metrics that the designer is optimizing for, i.e., the cost of the design decisions. The optimizer in turn considers the current design's cost, the system input description and constraints to output valid and optimized system design decisions.

Due to the high design complexity, it is a huge challenge to achieve high-fidelity yet fast cost and optimization functions. Conventional techniques for obtaining cost estimates, such as analytical models, often suffer from inaccuracies, while simulations are too time-consuming for frequent use in optimization processes. Similarly, traditional optimization approaches relying on heuristics

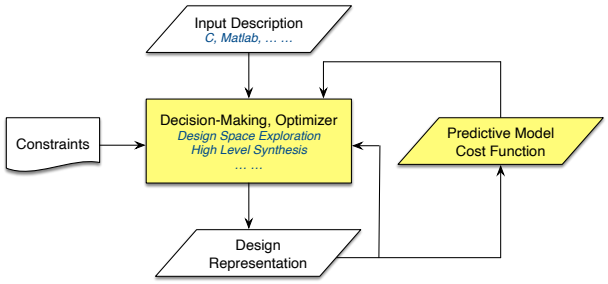

Figure 1: Embedded system design overview.

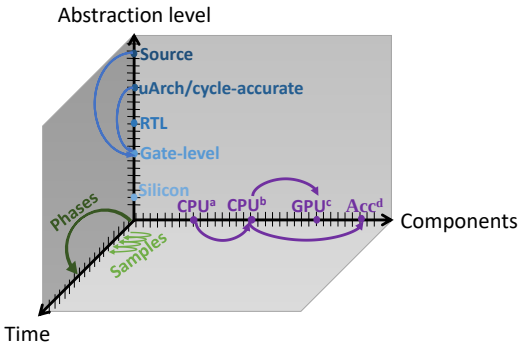

Figure 2: System-level predictive modeling axes [3].

or meta-heuristics for design space exploration suffer from sub-optimality or complexity inefficiencies. In contrast, machine learning (ML) techniques have the potential to overcome these challenges by leveraging data-driven learning and facilitating knowledge reuse. Moreover, ML-based knowledge reuse is much more flexible than IP block-based design reuse.

When applying ML to system design, either one or both of the decision-making and cost models are replaced with ML approaches. Supervised learning is commonly used to learn predictive cost models that replace inaccurate analytical or slow simulation methods. In terms of optimization, the framework depicted in Figure 1 naturally maps to a reinforcement learning (RL) methodology, with the decision-making being the RL agent or policy, the design representation being the state and the predictive model serving as the reward function, where RL-based optimization can again be constructive or incremental. Such RL-based optimization approaches have received a lot of interest at lower, e.g. physical levels of abstraction [41, 52], but corresponding work at higher abstraction levels is still an active area of research.

This paper presents a succinct review of the applications of ML techniques in embedded system design. Section 2 summarizes ML techniques for system-level modeling and design space exploration. Section 3 is focused on ML techniques for high-level synthesis (HLS). A review of ML-based register-transfer level (RTL) and logic synthesis prediction is provided in Section 4. Industrial perspectives on ML for embedded system design are introduced in Section 5. Finally, a summary and conclusions including open challenges and an outlook are offered in Section 6.

2 ML FOR SYSTEM-LEVEL DESIGN

The productivity of designing embedded systems can be significantly improved by raising the level of abstraction to encompass the entire system. At the system level, the focus is on mapping a behavioral design description onto a suitable platform made up of architectural components, such as processors, memories, hardware accelerators, and the interconnections between them [21]. At this level, fast yet accurate design and cost models are critical in making quick decisions and exploring the expansive design space within a constrained time budget. However, modeling efforts at such a high level of abstraction are particularly challenging. Analytical models are inherently limited due to their inability to model the many dynamic system effects. As such, system-level design (SLD) typically relies on simulations. At the same time, detailed simulations will severely hinder the exploration process due to their computational cost. To address these limitations, ML models have emerged as a promising solution to bridge the gap between fast yet inaccurate analytical models and slow but precise simulators. This section provides a survey of representative techniques in ML for system-level modeling. We also discuss various approaches for design space exploration (DSE) at the system level. Finally, we summarize our perspectives on open challenges in this field.

2.1 ML for System-Level Modeling

System-level modeling is concerned with delivering fast and accurate models of architectures and applications for system design, programming and runtime management. The work in applying ML for system-level modeling is large with diverse prediction targets and purposes, which can be categorized along three dimensions, represented in Figure 2 [3]. The dimensions comprise predictions across abstraction levels, across components, and across time.

Models that make **cross-layer predictions** learn the relationship between high-level features and low-level implementation properties such as performance, energy, reliability, power, thermal (PERPT), and others. Finding effective abstractions for high-level system modeling has received a lot of attention in the past [13, 22]. Traditionally, this involves back-annotating a fast functional high-level simulation with low-level PERPT estimates in a manual or automated fashion [84]. In cross-layer ML methods, the need to manually construct appropriately abstracted models is replaced with learning them instead. This can be done to learn high-level functional surrogate as well as non-functional PERPT models, but in this paper we focus on the latter. Figure 3 depicts an overview of the predictive PERPT modeling process with ML. In a supervised learning formulation, a high-level functional simulation is used to extract features that are correlated with low-level reference metrics to train a prediction model on selected micro-benchmarks. During inference, the learned model is attached to the high-level simulation to predict target metrics for previously unseen applications.

The design of predictive PERPT modeling techniques is an ongoing area of research [45]. Learning-based performance models have been successfully applied at higher levels without the need for any structural information [40]. Power modeling, on the other hand, relies on low-level details that are challenging to abstract in high-level models. Early works focused on simple regressions [63, 73], but suffer from capturing the inherently non-linear and data-dependent power characteristics at higher levels of abstraction. At the RTL deep learning (DL) approaches have been applied to predict power from an unfiltered union of all possible inputs [87] (see Section 4).

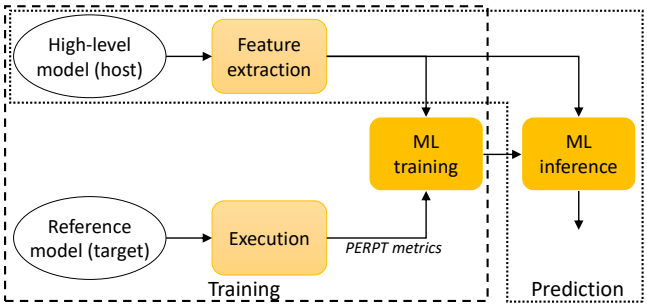

Figure 3: ML for system-level modeling overview.

However, DL incurs large training and inference overheads that negate the benefits of high-level modeling. At higher levels, approaches are needed that allow for much simpler learning formulations by leveraging and encoding a-priori knowledge, e.g., through feature selection that captures the known relationship between target metrics and functional activity instead of costly rediscovering it. The studies in [7, 8] proposed such an approach for learning-based power modeling of complex CPUs at the micro-architecture level. Similarly, at the source level, prior work extended functional simulations with the ability to predict power of hardware accelerators at cycle, basic block, or whole function granularity [30–32]. In all cases, results show that low-complex decision tree models can predict power with more than 95% accuracy requiring less than 30k training samples even for large designs.

When actual hardware components are available, we can replace reference models in Figure 3 with execution on a real target platform to effectively make **cross-platform predictions**. Such models learn and predict the relationship of program executions on different hardware platforms, typically using unintrusive hardware counter measurements obtained from running an unmodified application on a host as features to predict PERPT metrics on a different target. Some of the earliest work applied the concept of cross-platform prediction to develop approaches for accurate power and performance prediction across host and target CPUs with vastly different architectures [85, 86]. They use a locally constrained piecewise linear regression to predict the performance and power consumption of single-threaded programs executing on a single target core both at the whole program and program phase granularity (defined as a fixed number of basic blocks) with more than 95% accuracy. Similar cross-platform models have been constructed to make predictions between different graphic processing units (GPUs) as host and target [46], as well as CPU-to-GPU predictions [9, 10], and from CPU to FPGA [44]. Note that cross-layer predictions at the source level, CPU-to-FPGA cross-platform predictions and HLS cost predictions (see Section 3.1) share similar problem statements. The differences lie in the amount of feature extraction and target-specific design analysis used for predictions. Cross-platform models rely only on host-specific hardware counters as features. In general, they enable designers, programmers as well as runtime and operating systems to rapidly guide decisions about offloading and refactoring to partition and map applications across heterogeneous architectures [51].

Finally, **cross-temporal predictions** are aimed at allowing runtime systems to dynamically manage resources by exploiting fine-grained optimization opportunities that static runtime management would miss. It is known that workloads exhibit dynamic behaviors, where cross-temporal models learn and characterize dynamic workload patterns to predict future workload behavior. They enable systems to behave proactively by anticipating workload changes, adding the capability to pro-actively adapt to upcoming rather than only react to past behaviors. Predicting workload behaviors can be cast as a time series forecasting problem, where regression models are a typical solution. ML-based methods include linear predictors [57] to, more recently, DL [42] and recurrent neural networks [39, 55]. Their objective is to minimize the forecasting error of periodically measured metrics such as cycles per instruction (CPI). These models excel at predicting short-term upcoming workload behaviors, but struggle to predict abrupt long-term workload changes [2]. Such changes occur because workloads go through phases, where phase classification and prediction solutions have been extensively studied [5]. Predicting phases is a cross-temporal modeling approach that characterizes and forecasts long-term workload behaviors. However, it overlooks sample-by-sample variability. Recent studies proposed a phase-aware mechanism that combines long-term and short-term predictions to provide more accurate cross-temporal predictions [6], achieving up to 22% accuracy improvement over phase-unaware setups.

2.2 ML for System-Level Exploration

Predictive cost models are at the core of any SLD process. However, intelligently searching the system design space is another important requirement for finding optimal design solutions within a practical exploration time budget. This has led to the development of various methods. While early studies proposed simple, e.g. greedy heuristics [70], meta-heuristics such as multi-objective evolutionary algorithms coupled with simulation-based cost models are predominantly used for system-level DSE today [21].

More recently, RL approaches have demonstrated superiority against traditional meta-heuristics such as evolutionary algorithms. One of the advantages of RL models is that they are capable of generalizing their decisions, i.e. can learn from previous designs and apply such learned knowledge to new problem instances with similar characteristics. This makes RL an attractive option for system-level DSE. Recent examples include the use of Monte Carlo tree search (MCTS) to explore application-specific NoC design [25, 26] or Q-learning to optimize task allocation in MPSoCs [36].

Such RL formulations are capable of exploring the design space by themselves. Supervised learning can instead be applied to optimization problems, but relies on the designer to provide labeled ground truth, i.e. examples of optimal designs for a sufficient set of design instances and inputs. When the ground truth is readily available and design spaces are sufficiently easy to explore manually, supervised learning can be a powerful tool to generalize from unseen samples. Such scenarios typically arise when decisions are constrained to a single domain, e.g., voltage/frequency pair selection [14] or composite prefetcher throttling [4] in runtime systems.

2.3 Challenges and Opportunities

The application of ML methods for SLD presents several open challenges. A key challenge is the choice of appropriate learning formulations, with naive application of existing ML models often being a poor fit for system design problems. In particular, while DL

approaches have shown promise in various domains, their suitability for SLD is still a matter of consideration. SLD requires models that are not only accurate but also fast and efficient. DL models can be computationally expensive, and their training and inference times may exceed the time required for traditional low-level simulations, diminishing their practical utility. Furthermore, DL requires huge amounts of data for training, which is difficult to obtain at the system level. In scenarios where hardware is not yet available, collecting sufficient training data can be difficult. As outlined in this section, this calls for development of novel simple but effective learning formulations specifically for SLD.

Within this context, despite the advancements in automated feature engineering, human expertise still plays a vital role at the system level. Feature engineering helps reduce model complexity and training costs by identifying and incorporating relevant features into the learning formulation itself. However, striking a balance between involving human experts and leveraging automated feature engineering techniques remains a challenge.

Selecting training sets that adequately represent the diverse and complex behaviors of real-world workloads is generally a challenge. Designing training sets that generalize well across different system behaviors is crucial but challenging due to the vast design spaces. Some prior work has proposed generating synthetic data to overcome this challenge [48], but more research is needed.

Finally, SLD often requires integration of heterogeneous components from different vendors into system-on-chip (SoC) or other platforms. The absence of a centralized marketplace for models hinders the adoption and dissemination of ML models for SLD. The lack of a platform or repository where designers can access and share pre-trained models limits the collaboration and wider adoption of ML approaches in this field.

3 ML FOR HIGH-LEVEL SYNTHESIS

As modern embedded systems face rising complexity in both applications and hardware platforms, HSL has become a popular alternative to traditional RTL methodology to improve design productivity [15]. By raising the level of abstraction to untimed or partially timed design and providing a wide range of configuration options (e.g., pragmas), HLS facilitates fast exploration of various design alternatives. It achieves this by quickly generating RTL implementations for different configuration sets. This accelerated design iteration process empowers the designers to navigate through a vast design space in a more productive manner. However, the advantages of HLS come with a trade-off: it lacks detailed and crucial information from downstream implementation steps such as logic synthesis, placement, and routing. Consequently, HLS tools often produce resource and timing estimates that significantly deviate from the actual post-implementation quality-of-results (QoRs). This discrepancy poses a challenge for designers aiming to conduct effective design space exploration (DSE) in order to achieve desired QoRs. As a result, designers may need to invest a substantial amount of time in pushing the synthesized designs through subsequent stages of RTL synthesis and physical design. This trade-off offsets the benefits of HLS, despite its notable speed advantage over later steps in the implementation process [47].

Clearly, there is an inherent tension between speed and accuracy of QoR estimation in HLS-based design. Addressing this tension is crucial for achieving rapid design closure. To this end, recent research focuses on leveraging ML techniques in HLS to improve the quality of QoR estimation and reduce the need for extensive human supervision [27]. These efforts can be broadly categorized into the following two areas: (1) ML-based QoR estimation models: The primary obstacle to achieving efficient design closure lies in the correlation across different design stages. ML techniques, such as regression and classification models, have demonstrated their ability to narrow the QoR miscorrelation gap by providing faster and more precise estimation across stages; (2) Intelligent DSE with ML: Recent advancements in intelligent decision-making frameworks for DSE in HLS have shown promising results in achieving design closure. Techniques such as Bayesian optimization or reinforcement learning enable more informed decision-making, leading to improved design outcomes.

In the following, we provide a concise overview of the application of ML in HLS. We examine representative ML techniques utilized in the aforementioned research areas of QoR estimation and intelligent DSE. Additionally, we outline the key challenges associated with employing ML in HLS.

3.1 ML for High-Level QoR Estimation

Achieving accurate quality of results estimation at the HLS stage is challenging due to the cumulative effects of complex transformations across multiple implementation stages. To address the disparity between HLS estimation and actual QoR, prior studies in the literature have proposed various approaches, which can be categorized into three main targets: (1) resource usage estimation, (2) performance estimation (e.g., timing, latency), and (3) routing congestion estimation. These studies aim to bridge the gap and improve the accuracy of QoR estimation in HLS.

The authors in [17, 38] were among the first to utilize ML models, such as gradient boosting and artificial neural networks, to bridge the gap between resource usage estimated by HLS and the actual post-implementation results. They achieved this by extracting valuable features from the HLS report. While these approaches have been effective in reducing errors in resource estimation, extending them to timing estimation is difficult because the underlying ML models do not consider any structural features from the input design. To overcome this limitation, Ustun et al. [69] employ graph neural networks (GNNs) to learn operation mapping patterns for DSP and carry blocks on FPGAs, by incorporating structural information from the HLS intermediate representation (IR) to capture the relationships among operations. This mapping-awareness delay estimation, based on GNN predictions, significantly improves the accuracy of delay estimation in HLS. Subsequent studies, such as Sohrabizadeh et al. [61], Wu et al. [75], and Bai et al. [11], also leverage GNNs to predict timing and/or latency of the HLS design. These studies either make use of the HLS source code or its corresponding control data flow graph in the IR to enhance their predictions.

Another research direction focuses on leveraging ML for early prediction of routing congestion. Accurately detecting congested regions in HLS can enable effective resolution of potential timing closure issues during the early design stage. However, the lack of physical information hinders accurate predictions in HLS. To tackle this issue, Zhao et al. [83] propose an ML-based method to predict routing congestion in HLS and identify highly congested regions in

the source code. In their approach, the authors treat operators in the HLS IR as training samples and gather various features related to resource usage, operator types, and connections between operators. The ground truth data for operators is obtained by tracing back from the routing congestion metrics observed during placement and routing. Using these features and ground truth data, a gradient boosted regression tree model is trained to predict the congestion metrics for operators at the HLS stage.

3.2 ML-Aided High-Level Design Exploration

In the context of HLS, the DSE problem entails finding the optimal configuration of parameters, such as loop unrolling factors, pipelining initiation intervals (IIs) or the number of array partitions, which are typically specified using pragmas or directives as part of the high-level specification. The design space encompasses the complete range of possible design configurations. Traditionally, DSE in HLS has been approached using analytical or genetic algorithm-based methods [58, 82]. However, these methods often face scalability and exploration efficiency challenges.

In recent years, ML-based techniques such as active learning and Bayesian optimization have emerged as promising approaches for HLS DSE. Active learning [79], focuses on selecting representative and challenging samples from the design space to maximize the predictive model's accuracy using minimal training samples [33]. On the other hand, Gaussian processes and Bayesian optimization [35, 62] employ probabilistic models to predict the performance of unseen design points and iteratively refine the model.

Similar to QoR estimation, incorporating structural information can enhance the efficiency and effectiveness of the DSE process. Ferretti et al. [19] employ GNNs built on control dataflow graphs annotated with HLS directives. These GNN models act as performance predictors, efficiently transferring learned knowledge to new tasks. By leveraging the structural information captured by GNNs, performance prediction becomes more accurate and efficient. Another approach, as presented in [74], combines GNNs with reinforcement learning. Their method, called IronMan, predicts performance using the original dataflow graph of the input program, encompassing both regular and irregular data paths. RL is then utilized to explore optimal resource allocation strategies based on user-specified constraints. Transfer learning has also been leveraged to enhance the efficiency and effectiveness of the HLS DSE process. For instance, in [28], authors utilize knowledge acquired from related design spaces to accelerate the exploration of new HLS programs. These more sophisticated ML methods demonstrate the potential benefits of utilizing structural information and data-driven techniques in HLS DSE, leading to improved results.

3.3 Challenges and Potential Solutions

Although significant progress has been made in applying machine learning to HLS, there are still several challenges that need to be addressed. Here, we highlight a few of these challenges.

Similar to the system level, ML algorithms for HLS heavily rely on data to learn patterns and make accurate predictions, where collecting sufficient and diverse training data can be challenging due to the complexity and specificity of hardware design tasks. As a result, models may suffer from poor training or limited transferability, leading to suboptimal performance on unseen data or new tasks. In their work, Ustun et al. [69] propose a potential solution to this challenge by generating synthetic HLS designs to cover a wide range of data patterns. We believe that similar data augmentation techniques and new approaches, particularly those leveraging advancements in large language models, can help alleviate the scarcity of labeled HLS designs.

In addition, achieving timing closure in HLS designs remains a significant challenge that hinders productivity. While there have been several studies focusing on timing and congestion estimation in HLS designs, accurately predicting critical path delay or clock frequency is a nontrivial task due to its inherent complexity. Moreover, even if an ML model can accurately estimate the delay of the critical path, guiding the tool to effectively resolve the timing issue remains difficult for designers [16, 29]. Recent efforts have shown promising results in improving the clock frequency of HLS designs by combining a coarse-grained floorplanning step with pipelining during the HLS compilation process [23, 24]. We believe that further enhancements can be achieved by integrating ML predictions into this strategy to accelerate the design closure process.

4 RTL/LOGIC SYNTHESIS PREDICTION

Modern embedded computing systems employ accelerator-rich SoCs to meet new computation demands. Unlike general-purpose processors that are programmable, accelerators sacrifice generality for specificity to offer unmatched performance and efficiency. However, accelerators face friction when being adopted by industry on a large scale because they are time-consuming to develop and difficult to program. The only way accelerators can keep up with the ever-changing computation demands is by having a fast development process and a relatively short design iteration cycle.

4.1 ML-Aided Modeling of Hardware Costs

To expediently and accurately model the hardware design cost of an RTL circuit, such as an accelerator's power, area, and timing, a myriad of ML-aided techniques have been proposed. AutoAx [43] builds upon traditional synthesis tools and aims to accelerate the entire synthesis flow by employing libraries of approximate components. This strategy is designed to expedite the design space exploration of approximate circuits. However, it is not general-purpose enough to predict the cost of any arbitrary RTL circuit design. Others have proposed utilizing ML models to quickly predict the power of an RTL design. Apollo [77] utilizes a linear model that can act as an on-chip power meter and guide the dynamic voltage and frequency scaling of microprocessors. Both PRIMAL [87] and PowerNet [76] employ convolutional neural networks (CNNs) as their models, and GRANNITE [81], on the other hand, utilizes GNNs that operate directly on the graph representation of RTL circuits. PowerNet predicts the runtime voltage drop of a hardware design, while PRIMAL and GRANNITE predict the runtime power of a given circuit design using input application execution traces. For example, given the layout of a particular general-purpose processor, PRIMAL and GRANNITE can estimate the runtime power consumption of applications. All three of these models show substantial speedup over traditional synthesis tools.

However, these machine learning models have limitations. First, some of them are trained on data from lower abstraction levels than designs written in RTL or other hardware description languages.

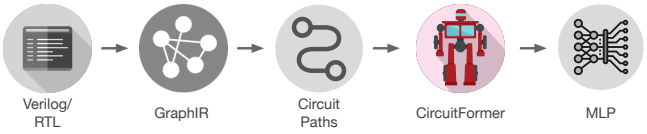

Figure 4: SNS takes Verilog/RTL codes of a hardware design as input, turns the RTL codes into a graph intermediate representation (GraphIR), samples circuit paths from the GraphIR, uses an augmented super light-weight Transformer model called CircuitFormer to predict path-based attributes, and finally aggregates path-based information into predicted physical characteristics (i.e., area, power, timing) of the hardware design in its entirety.

For example, PRIMAL and PowerNet operate at the layout level, and GRANNITE operates at the logic gate level. The process of translating RTL circuit designs from HDL-level abstraction to lower-level abstractions such as layout is time-consuming since synthesis and place-and-route need to be performed. These ML models are better suited for estimating application power consumption of a handful of designs but not suitable to perform hardware design space explorations where hundreds and thousands or more designs are explored. Second, most models only predict hardware costs for a limited set of designs or only predict a particular hardware design cost. For example, Apollo only functions with a few select microprocessor core implementations. PowerNet, PRIMAL, and GRANNITE are only capable of predicting the power consumption of RTL circuit designs, which is only a component of the hardware design costs. Finally, some models do not scale well with large input designs or simply cannot predict the hardware cost of large input designs. For example, PRIMAL and GRANNITE encounter a significant slowdown when predicting larger designs compared to predicting smaller ones. The largest input design demonstrated is only a small RISC-V core (<1 mm^2 without caches). The capacity to scale to large designs is crucial in the realm of modern embedded systems modeling. This is particularly significant as the scale and complexity of modern embedded systems continue to rise, with a single SoC now housing dozens of heterogeneous cores.

4.2 Path-Based Approach Enables Scalable Hardware Cost Prediction

Recent work SNS [78] leverages advances in artificial intelligence and graph analytics to turn input hardware designs into graph representations for synthesis predictions. Figure 4 shows the flow. SNS takes a novel circuit-path-based approach to predict the physical characteristics of individual circuit paths and aggregating the paths' characteristics to predict the area, power, and timing of the entire input design. By exploiting sequence processing techniques that learn the order and the placement of words in relation to a sentence such as in natural language processing (NLP) [71], SNS learns the order and the placement of functional units in a circuit path to provide more accurate synthesis result predictions. With a very limited number of open-source hardware designs available as training data, SNS utilizes generative models [80] to generate training datasets, providing accurate predictions even when training data is scarce.

Rather than working on the graph representation of the entire design, the path-based approach provides several advantages over

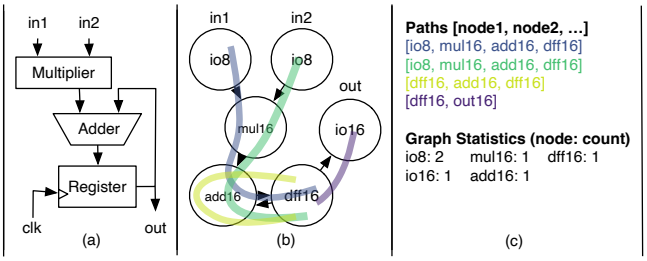

Figure 5: Extracting path-based physical characteristics.

prior work. First, circuit paths are less compute-intensive to synthesize for "ground truths" when generating training data. Second, circuit paths are easier to process and faster to infer compared to entire design graphs, providing a scalable solution for predicting large input designs. Third, the path-based approach allows ML models to infer "local properties" in the design with high accuracies, such as utilizing critical paths of a design to infer timing. This prevents interference from adjacent, unrelated hardware modules. In addition, the ability to pinpoint the exact path that contributes to the critical path, for example, allows for specific optimization of the design that is otherwise unattainable. Last but not least, circuit paths can be generated using a GAN-base method as described below, combating ML model training data scarcity. Figure 5 shows pictorially how SNS turns an 8-bit multiply-add unit in (a) into the circuit graph in (b) and extracts the circuit paths and graph statistics such as the counts of each distinct node name as shown in (c).

Predicting the physical characteristics of an input design using all circuit paths of the design, or exhaustively sampling all circuit paths, is unnecessary because lots of circuit paths are similar and do not give us additional information. Recent work [49] demonstrates that performing random path sampling over the entire network graph and analyzing these sampled paths provide sufficient graph information to perform graph analysis. SNS takes a similar approach and uses a depth-first-search-based algorithm to randomly sample circuit paths across the entire design. Instead of randomly sampling any circuit paths, hardware design domain knowledge is incorporated so that only circuit paths beginning and ending with flip-flops are sampled, vastly reducing the search space. These sampled paths essentially capture the "one-cycle behavior" of the design, making it possible to predict timing (i.e., the critical path) of the circuit paths and therefore timing of the entire design.

4.3 Exploiting NLP Model and GAN to Provide More Accurate Hardware Cost Prediction

Traditional synthesis tools such as the Synopsys Design Compiler (DC) employ optimizations to create a synthesized design that exhibits smaller area, power, and better timing whenever possible. For example, an adder followed by a multiplier will be inferred as a multiply-accumulate unit. This example illustrates the importance of the ordering and placement of nodes that represent functional units, registers, and flip-flops on a circuit path. To improve the accuracy of hardware cost predictions, a stripped-down and augmented Transformer model [71], the most popular NLP model, is used. This model takes into account the ordering and the placement of the tokens (nodes) to predict the circuit path's physical characteristics with exceptional speed and accuracy. With

high-level features extracted from individual circuit paths via the mini-Transformer model, multi-level perceptrons are used to aggregate path-level inferences to predict the area, power, and timing of the entire hardware design. For example, the timing prediction of a hardware design is the maximum of the sampled circuit paths' timing predictions.

Besides leveraging an effective model to predict hardware costs, sufficient training data is also necessary to achieve desired prediction accuracy. Unfortunately, good quality open-sourced RTL designs are scarce. This makes training a deep learning model with "ground truths" challenging. In addition, successfully synthesizing these RTL designs often require human effort, and once the synthesis flow is bug-free, it is computationally expensive and time-consuming to generate synthesis results.

To create a sufficiently large training dataset, SNS extracts circuit paths directly from the available hardware designs as well as generates new, artificial yet realistic circuit paths to combat data scarcity. Two specific techniques are employed: Markov Chain and SeqGAN (Generative Adversarial Network for sequences) methods. The former is based on a transition matrix of conditional probabilities and is used to generate unique, realistic circuit paths. The latter is adept at creating longer, meaningful paths by learning the sequence of nodes from actual hardware designs. This combination ensures that the data generation and augmentation processes produce diverse, less biased, and yet realistic circuit paths. As the final step in creating the training dataset, all generated circuit paths are synthesized using Synopsys DC to obtain real synthesis results as the training targets. This approach, taking advantage of traditional synthesis toolchain feedback and ML generative methods, ensures a robust and accurate model in the presence of limited original data.

4.4 Challenges and Opportunities

Data scarcity remains a critical challenge despite the advances in using GANs to generate a diverse, unbiased training dataset. Compromises were made when encoding wires and functional unit bit-widths to keep the training vocabulary set size reasonable for a mini-Transformer model to ensure feasible data augmentation and model scalability. For example, only power-of-two bit-widths are supported, e.g., a 29-bit adder or a 34-bit adder would be "rounded-near" to a 32-bit adder. Future work can address this inaccuracy by developing a more robust encoding scheme without increasing the model size significantly.

The limited transferability of the prediction model is another critical challenge. For example, if the model is trained on a 32nm technology library, the model can successfully predict the synthesis results of designs it has never seen before, only if the designs use the same 32nm technology library. However, the model will need to be retrained from scratch to predict designs that are implemented in a different technology node or operating under a different condition (e.g., different operating corners). Since the process of collecting and generating the requisite dataset is resource-intensive and time-consuming, this restricts the practical applicability of these models. Future research can address this challenge by developing a more transferable model without requiring complete retraining of the model for predicting hardware design costs of implementations in alternate technology libraries or operating conditions.

5 INDUSTRY PERSPECTIVES

At Infineon, we recognize the high potential of ML to accelerate and improve our industrial design flow. The primary enablers of ML applications are Electronic Design Automation (EDA) tools and modern ML techniques. First, EDA tools generate a considerable amount of objects, reports, and log files that could be used for ML training. Data availability is especially notorious in the industry, where each subteam across the design flow develops different IPs and runs different EDA tools multiple times. Besides, collection systems for relevant data metrics are available so each run's results can be stored, tracked, and analyzed. Second, modern ML techniques work over different data structures, such as vectors, matrices, and graphs. The latter represents an excellent chance for research in the field, as EDA objects and tasks are naturally represented with graph structures. In addition, we also observe that ML finds its way more and more through EDA tools to improve their capabilities and simplify their usage. Nevertheless, there are still many open additional ML applications, mainly in the area of design flow (i.e., involvement of a set of tools) and beyond RTL.

As RTL design is the entry point of the design flow, its quality impacts the design results after physical synthesis. RTL variants using different micro-architectures and HDL coding styles have a different impact on Power, Performance, and Area (PPA) results [60]. Thus, optimizations should be made at RTL.At Infineon, we believe that providing a right-at-first RTL would reduce iterations of the digital design flow. However, finding a right-at-first design implies measuring PPA values from different RTL variants. This measure is costly and time-consuming, as logical and physical synthesis tools should be run. To tackle this and enable RTL design exploration, accurate yet fast estimations of the PPA of a design are needed.

As timing closure is a crucial aspect of a design, we use ML to provide a fast estimation of the timing behavior of the components of a digital circuit at the early stages of the design flow. In particular, we use ML models inside our in-house RTL generation framework to estimate pin-to-pin delays of its primitive components. The building blocks of our ML solution are the data collection flow for getting training and evaluation circuits and the task and architecture definition to correctly map the EDA task as an ML problem and choose a suitable architecture.

5.1 Data Collection Flow

The widespread use of embedded systems implemented in continuous shrinking technologies results in more strict requirements for the design flow. To cope with this, Infineon has developed an RTL generation framework called MetaRTL [59]. MetaRTL uses meta-modeling and applies the Model Driven Architecture (MDA) principle to RTL generation [68]. This framework provides three layers of a design: the initial specifications or Model of Things (MoT), the intermediate representation of the design's micro-architecture or Model of Design (MoD), and the mapping to an HDL or Model of View (MoV). MetaRTL generates different RISC-V-based CPUs, accelerators, IPs, and SoCs while reusing generators and saving design and verification efforts [18, 20, 50].

MetaRTL's generated RTL designs are synthesized to obtain the QoRs of the design in terms of PPA. To that end, we incorporate commercial and open-source physical synthesis tools into our design flow, going from the specification level to the final GDS layout.

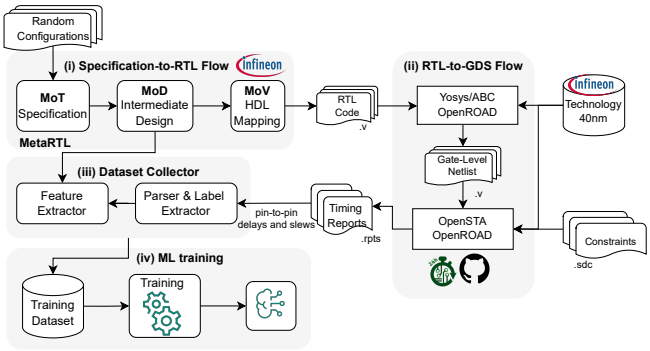

Figure 6: Industrial data collection flow.

Specifically, we use Yosys [72] for logic synthesis, and the OpenROAD [1] toolchain for physical synthesis and timing analysis. In [37], we present the challenges, solutions, and opportunities of using OpenROAD in our industrial flow and with our proprietary technologies. In [66], PPA results derived from OpenROAD are compared w.r.t. commercial tool results over different RISC-V-based CPU cores generated by MetaRTL. Even though commercial tools outperform OpenROAD results for the same design configuration and constraints, our data collection flow in Figure 6 leverages open-source tools for transparency on the synthesis flow, saving license costs and promoting research in the field.

The advantages of our data collection flow are threefold. First, training and evaluation designs can be easily generated by changing the specifications or the implemented micro-architecture at RTL. Second, formal properties are generated, and it is possible to verify MetaRTL designs following a four-eyes principle [18]. Finally, Python is used as the programming language for both tasks, hardware generation, and ML applications. Nevertheless, MetaRTL's resulting micro-architectures are not optimal since the RTL abstraction does not regard timing. Instead, MetaRTL relies entirely on the engineer coding the generator to implement timing-efficient micro-architectures. To tackle this, MetaRTL would require early knowledge about the timing characteristics of the generated designs. However, this timing information is computationally expensive to gather, as runs of synthesis and timing analysis tools are required.

5.2 Task and Architecture Definition

As timing is the most crucial design aspect, ML models are being used to estimate timing-related design metrics at different stages of the design flow [54]. However, most works train ML models using features from gate-level netlists, losing the mapping to the design at RTL. Other works, such as [60], map the Verilog code to an abstract syntax tree representation and train ML models to predict a design's post-placement total negative slack and power consumption. Similarly, we aim to predict timing metrics from the RTL design phase. We incorporate ML models into the RTL generation framework used at Infineon. The ML solution should be accurate, fast, and generalizable to any circuit. Thus, we aim to learn the timing behavior of each primitive component in the MoD or intermediate layer of the generation framework. Finally, the ML solution should be aware of the problem, i.e., features and model architecture must match the nature of the task.

In [64], we train vector-based ML models to predict the pin-to-pin delay of components within an intermediate RTL representation. The training data are collected following the flow in Figure 6. Small designs are generated using random configurations of primitive components such as logic gates, in a bitwise, logical, and reduced fashion. These designs are hierarchically synthesized, and timing analysis is run to collect timing reports. The latter are parsed to extract pin-to-pin delays and slew values. The features give information about the design configuration and are selected based on the physical models for delay calculation. Specifically, non-linear delay models are described in technology libraries and defined as lookup tables indexed by the output slew or transition time and the load capacitance. Similarly, the output slew is a function of the load capacitance and the propagated input slew. Thus, selected features include the component type, number of inputs, current pin, bit widths, and fan out. Deep ML models are trained to predict the pin-to-pin delay of logic components. Results show how results improve when considering the input slew as a feature and the output slew as an objective. Moreover, the model inference on larger evaluation circuits is made sequentially to consider that the slew is propagated along the paths of the timing graph, i.e., the output slew of a component is the input slew of the next component in the timing graph.

As circuits are built not only from logic components, the dataset is enlarged to cover more component types, such as arithmetic, branch, and multiplexer operators [65]. Even though more complex circuits could be evaluated, considering one model to predict the timing behavior of all component types decreases the model performance, even when considering slews as a feature. Further analysis shows that training one model to cover all training data is inappropriate, as different component types have different structural features and timing behaviors. Finally, inspired by the recent advances in GNNs and their numerous applications to the digital design flow [56], we employ GNNs to solve the pin-to-pin delay prediction task [67]. The same data collection flow in Figure 6 is used, but instead of building tabular datasets, we build a set of graphs representing the design at the intermediate RTL representation layer. Using GNNs for logic components provides better results than vector-based ML models, as graphs better represent our designs than the tabular pin-to-pin dataset. Moreover, timing analysis uses a timing graph to propagate delay and slews along the paths.

We have seen that ML provides a fast yet accurate estimation of component delays without running synthesis tools. Experimental results show that GNN-based models are promising. In future work, they can be employed to generalize to more component types supported by MetaRTL. Finally, ML-estimated delays could be used to detect critical blocks of the design so that timing violations can be identified earlier and better RTL micro-architectures can be found.

5.3 Open Challenges and Opportunities

EDA tools have been developed to cover the design flow from different levels of abstraction, from lumped-element models to the system specification. ML approaches allow abstracting EDA tasks even more. By learning from pattern distributions in a massive training data set, ML models reduce the complexity of the EDA tasks representing waveforms, circuits, and layouts in higher or lower-dimensionality vectors. However, in contrast to classical

abstractions, it is still not possible to understand what is being abstracted by an ML model. Thus, ML-based predictions cannot be fully explained or trusted (also w.r.t. adversarial samples [34]). This lack of explainability or interpretability lowers the acceptance of ML-based solutions replacing classical methods in industrial flows.

Additionally, ML models still cannot solve complex EDA tasks without restricting the problem to a smaller subspace. This solution space is as limited as the collected training data are, as data can be biased or suffer from distribution shifts. Even with a proper data collection flow allowing different RTL variants and collecting the PPA results of each, no ML model can be trained to cover all ranges of features, all technology nodes, all possible target clock frequencies, or all possible design rules and constraints. According to this, our ML-based timing estimation has provided sufficiently accurate results when restricting the problem to a specific timing setup, component types, and technology node and when considering the underlying physical models of the task, i.e., adding slew estimation and propagation.

The familiar aphorism *"All models are wrong, but some are useful"* from the statistician George E.P. Box [12] alludes to the limitations of any abstraction model w.r.t. the complexity of real problems and how these models can still be helpful. In this sense, all ML models are wrong too. Even though ML is an alternative for mapping EDA tasks to a space of reduced complexity and still getting significantly good results, ML models still need to improve on solving complete complex EDA tasks [53]. However, the aphorism from George Box also states that some models are useful—especially when adequately handled. In this sense, our ML-based time estimation is an excellent help to come up with a good architecture early. Furthermore, we explore using the data to do early trade-off analysis and optimization tasks automatically.

Future research in ML for embedded system design should focus on using ML with classical optimization and heuristic algorithms in a coarse or fine-grain integration, as proposed in [53]. The authors propose solving EDA tasks using ML models and classical algorithms side by side or one inside the other. This will help to solve more complex tasks without restricting the solution space and achieve better performance. Moreover, ML-based solutions should not be black-box solutions. They should instead be aware of the physical models of the underlying task to select meaningful features and patterns from the training data. These two basic directions can open the door to reliable ML models solving core EDA tasks across different levels of abstraction.

6 SUMMARY, CONCLUSIONS AND OUTLOOK

The recent efforts summarized in this paper have demonstrated the high potential of ML techniques in revolutionizing embedded system design and overcoming existing challenges. Supervised learning techniques provide high-fidelity and fast predictive models, which play a critical role in efficiently guiding solution search in huge and complicated design space. The reinforcement learning framework can be directly leveraged for design optimizations.

However, ML for embedded system design is still far from being sufficiently studied and there are new challenges associated with the ML techniques. First, the effectiveness of ML techniques heavily relies on a large volume of well curated data. However, a common theme across all abstraction levels is that obtaining labeled training data for embedded system designs is time consuming and computing intensive process. Within this context, the redundant nature of each research and development team independently undertaking expensive data preparation efforts leads to significant computational waste. Moreover, when there are technological or application changes, reusing existing data is still difficult, necessitating a restart of the data preparation process. Consequently, data efficiency becomes a crucial challenge and bottleneck impeding the advancement of ML-based embedded system design.

Another challenge is the interoperability between ML techniques and human manual design. In system-level designs, human manual intervention remains indispensable due to the inherent value of human design knowledge and the current limitations of system-level design automation techniques. However, there is no streamlined approach to incorporate human design knowledge into ML techniques other than design specifications, trial-and-error cost functions or labor-intensive manual feature engineering. Conversely, the limited interpretability of ML techniques poses a challenge for designers seeking to integrate ML solutions with human decision-making processes.

To address the aforementioned challenges, we propose the following potential solutions. To overcome the data efficiency challenge, we advocate for the establishment of shared data and machine learning models in embedded system designs, taking inspiration from successful approaches utilized by ImageNet and Kaggle within the machine learning community. Achieving this goal will require collaborative efforts from the entire embedded system community. Additionally, for the challenge of interoperability between human design and ML techniques, beyond more research into learning formulations, leveraging the recent advancements in large language models has the potential to significantly advance the seamless incorporation of human design knowledge with ML techniques.

ACKNOWLEDGMENTS

This work is partially supported by NSF grants CCF-1763848 and CCF-2212346, as well as the German Federal Ministry for Economic Affairs and Energy (BMWi) as part of the research project ProgressivKI (19A21006C).

REFERENCES

[1] T. Ajayi et al. 2019. INVITED: Toward an Open-Source Digital Flow: First Learnings from the OpenROAD Project. In *DAC*.
[2] E. S. Alcorta et al. 2021. Phase-Aware CPU Workload Forecasting. In *SAMOS*.
[3] E. S. Alcorta et al. 2022. Machine Learning for System-Level Modeling. In *Machine Learning Applications in Electronic Design Automation*, Haoxing Ren and Jiang Hu (Eds.). Springer, 545–579.
[4] E. S. Alcorta et al. 2023. Lightweight ML-based Runtime Prefetcher Selection on Many-core Platforms. In *MLArchSys*.
[5] E. S. Alcorta and A. Gerstlauer. 2021. Learning-Based Workload Phase Classification and Prediction Using Performance Monitoring Counters. In *MLCAD*.
[6] E. S. Alcorta and A. Gerstlauer. 2022. Learning-based Phase-aware Multi-core CPU Workload Forecasting. *ACM TODAES* 28, 2 (2022), 23:1–23:27.
[7] A. K. Ananda Kumar et al. 2022. Machine Learning-Based Microarchitecture-Level Power Modeling of CPUs. *IEEE TC* 72, 4 (2022), 941–961.
[8] A. K. Ananda Kumar and A. Gerstlauer. 2019. Learning-Based CPU Power Modeling. In *MLCAD*.
[9] N. Ardalani et al. 2015. Cross-architecture performance prediction (XAPP) using CPU code to predict GPU performance. In *MICRO*.
[10] N. Ardalani et al. 2019. A Static Analysis-based Cross-Architecture Performance Prediction Using Machine Learning. arXiv:1906.07840
[11] Y. Bai et al. 2023. ProgSG: Cross-Modality Representation Learning for Programs in Electronic Design Automation. arXiv:2305.10838

[12] G. EP Box and N. R Draper. 1987. *Empirical model-building and response surfaces*. John Wiley & Sons.
[13] O. Bringmann et al. 2015. The Next Generation of Virtual Prototyping: Ultra-fast Yet Accurate Simulation of HW/SW Systems. In *DATE*.
[14] R. Cochran et al. 2011. Pack & Cap: adaptive DVFS and thread packing under power caps. In *MICRO*.
[15] J. Cong et al. 2011. High-Level Synthesis for FPGAs: From Prototyping to Deployment. *IEEE TCAD* 30, 4 (2011), 473–491.
[16] J. Cong et al. 2022. FPGA HLS today: successes, challenges, and opportunities. *ACM TRETS* 15, 4 (2022), 1–42.
[17] S. Dai et al. 2018. Fast and accurate estimation of quality of results in high-level synthesis with machine learning. In *FCCM*.
[18] K. Devarajegowda et al. 2019. How to Keep 4-Eyes Principle in a Design and Property Generation Flow. In *MBMV*.
[19] L. Ferretti et al. 2022. Graph Neural Networks for High-Level Synthesis Design Space Exploration. *ACM TODAES* 28, 2 (2022), 1–20.
[20] N. Gerlin et al. 2022. Design of a Tightly-Coupled RISC-V Physical Memory Protection Unit for Online Error Detection. In *VLSI-SoC*.
[21] A. Gerstlauer et al. 2009. Electronic System-Level Synthesis Methodologies. *IEEE TCAD* 28, 10 (2009), 1517–1530.
[22] A. Gerstlauer et al. 2012. Abstract System-Level Models for Early Performance and Power Exploration. In *ASP-DAC*.
[23] L. Guo et al. 2021. AutoBridge: Coupling Coarse-Grained Floorplanning and Pipelining for High-Frequency HLS Design on Multi-Die FPGAs. In *FPGA*.
[24] L. Guo et al. 2022. RapidStream: Parallel Physical Implementation of FPGA HLS Designs. In *FPGA*.
[25] Y. Hu et al. 2018. Wavefront-MCTS: Multi-objective Design Space Exploration of NoC Architectures based on Monte Carlo Tree Search. In *ICCAD*.
[26] Y. Hu et al. 2020. Machine Learning Approaches for Efficient Design Space Exploration of Application-Specific NoCs. *ACM TODAES* 25, 5 (2020), 44:1–44:27.
[27] G. Huang et al. 2021. Machine learning for electronic design automation: A survey. *ACM TODAES* 26, 5 (2021), 1–46.
[28] J. Kwon and L. P Carloni. 2020. Transfer learning for design-space exploration with high-level synthesis. In *MLCAD*.
[29] Y. Lai et al. 2021. Programming and Synthesis for Software-Defined FPGA Acceleration: Status and Future Prospects. *ACM TRETS* 14, 4 (2021), 1–39.
[30] D. Lee et al. 2015. Dynamic Power and Performance Back-Annotation for Fast and Accurate Functional Hardware Simulation. In *DATE*.
[31] D. Lee et al. 2015. Learning-Based Power Modeling of System-Level Black-Box IPs. In *ICCAD*.
[32] D. Lee and A. Gerstlauer. 2018. Learning-Based, Fine-Grain Power Modeling of System-Level Hardware IPs. *ACM TODAES* 23, 3 (2018), 30:1–30:25.
[33] H. Liu and L. P Carloni. 2013. On learning-based methods for design-space exploration with high-level synthesis. In *DAC*.
[34] K. Liu et al. 2021. Can We Trust Machine Learning for Electronic Design Automation?. In *SOCC*.
[35] C. Lo and P. Chow. 2018. Multi-fidelity optimization for high-level synthesis directives. In *FPL*.
[36] S. Lu et al. 2015. Reinforcement Learning for Thermal-aware Many-core Task Allocation. In *GLSVLSI*.
[37] C. Lück et al. 2022. Industrial Experience with Open-Source EDA Tools. In *MLCAD*.
[38] H. Makrani et al. 2019. Pyramid: Machine learning framework to estimate the optimal timing and resource usage of a high-level synthesis design. In *FPL*.
[39] D. Masouros et al. 2021. Rusty: Runtime Interference-Aware Predictive Monitoring for Modern Multi-Tenant Systems. *IEEE TPDS* 32, 1 (2021), 184–198.
[40] C. Mendis et al. 2019. Ithemal: Accurate, Portable and Fast Basic Block Throughput Estimation using Deep Neural Networks. In *ICML*.
[41] A. Mirhoseini et al. 2020. Chip Placement with Deep Reinforcement Learning. arXiv:2004.10746
[42] M. Moghaddam et al. 2018. Dynamic Energy Optimization in Chip Multiprocessors Using Deep Neural Networks. *IEEE TMSCS* 4, 4 (2018), 649–661.
[43] V. Mrazek et al. 2019. autoax: An automatic design space exploration and circuit building methodology utilizing libraries of approximate components. In *DAC*.
[44] K. O'Neal et al. 2018. HLSPredict: cross platform performance prediction for FPGA high-level synthesis. In *ICCAD*.
[45] K. O'Neal and P. Brisk. 2018. Predictive Modeling for CPU, GPU, and FPGA Performance and Power Consumption: A Survey. In *ISVLSI*.
[46] K. O'Neal et al. 2019. Hardware-Assisted Cross-Generation Prediction of GPUs Under Design. *IEEE TCAD* 38, 6 (2019), 1133–1146.
[47] D. Pal et al. 2022. Machine Learning for Agile FPGA Design. In *Machine Learning Applications in Electronic Design Automation*, Haoxing Ren and Jiang Hu (Eds.). Springer, 471–504.
[48] R. Panda et al. 2016. Genesys: Automatically Generating Representative Training Sets for Predictive Benchmarking. In *SAMOS*.
[49] B. Perozzi et al. 2014. DeepWalk. In *SIGKDD*.
[50] S. Prebeck et al. 2022. A Scalable, Configurable and Programmable Vector Dot-Product Unit for Edge AI. In *MBMV*.
[51] A. Prodromou et al. 2019. Platform-Agnostic Learning-Based Scheduling. In *SAMOS*.
[52] M. Rapp et al. 2022. MLCAD: A Survey of Research in Machine Learning for CAD Keynote Paper. *IEEE TCAD* 41, 10 (2022), 3162–3181.
[53] H. Ren et al. 2023. Machine Learning and Algorithms: Let Us Team Up for EDA. *IEEE Design & Test* 40, 1 (2023), 70–76.
[54] H. Ren and J. Hu. 2023. *Machine Learning Applications in Electronic Design Automation*. Springer.
[55] M. Sagi et al. 2021. Long Short-Term Memory Neural Network-based Power Forecasting of Multi-Core Processors. In *DATE*.
[56] D. Sánchez et al. 2023. A Comprehensive Survey on Electronic Design Automation and Graph Neural Networks: Theory and Applications. *ACM TODAES* 28, 2 (2023), 1–27.
[57] R. Sarikaya and A. Buyuktosunoglu. 2007. Predicting Program Behavior Based On Objective Function Minimization. In *IISWC*.
[58] B. C. Schafer and K. Wakabayashi. 2012. Divide and Conquer High-Level Synthesis Design Space Exploration. *ACM TODAES* 17, 3 (2012), 1–19.
[59] J. Schreiner et al. 2016. Design centric modeling of digital hardware. In *HLDVT*.
[60] P. Sengupta et al. 2022. How Good Is Your Verilog RTL Code? A Quick Answer from Machine Learning. In *ICCAD*.
[61] A. Sohrabizadeh et al. 2022. Automated Accelerator Optimization Aided by Graph Neural Networks. In *DAC*.
[62] Q. Sun et al. 2022. Correlated multi-objective multi-fidelity optimization for HLS directives design. *ACM TODAES* (2022), 46–51.
[63] D. Sunwoo et al. 2010. PrEsto: An FPGA-accelerated Power Estimation Methodology for Complex Systems. In *FPL*.
[64] D. Sánchez Lopera et al. 2021. RTL Delay Prediction Using Neural Networks. In *NorCAS*.
[65] D. Sánchez Lopera et al. 2022. Early RTL delay prediction using neural networks. *Elsevier MICPRO* 94 (2022), 104671.
[66] D. Sánchez Lopera et al. 2022. Using Open-Source EDA Tools in an Industrial Design Flow. In *DVCON*.
[67] D. Sánchez Lopera and W. Ecker. 2022. Applying GNNs to Timing Estimation at RTL. In *ICCAD*.
[68] F. Truyen. 2006. The fast guide to model driven architecture. *Cephas Consulting Corp* (2006).
[69] E. Ustun et al. 2020. Accurate operation delay prediction for FPGA HLS using graph neural networks. In *ICCAD*.
[70] F. Vahid and T. Givargis. 2002. *Embedded System Design: A Unified Hardware/Software Introduction*. Wiley.
[71] A. Vaswani et al. 2017. Attention Is All You Need. arXiv:1706.03762
[72] C. Wolf et al. 2013. Yosys- A free Verilog synthesis suite. In *Austrochip*.
[73] G. Wu et al. 2015. GPGPU performance and power estimation using machine learning. In *HPCA*.
[74] N. Wu et al. 2021. Ironman: GNN-assisted design space exploration in high-level synthesis via reinforcement learning. In *GLSLSI*.
[75] N. Wu et al. 2022. High-level synthesis performance prediction using GNNs: benchmarking, modeling, and advancing. In *DAC*.
[76] Z. Xie et al. 2020. PowerNet: Transferable dynamic IR drop estimation via maximum convolutional neural network. In *ASP-DAC*.
[77] Z. Xie et al. 2021. APOLLO: An Automated Power Modeling Framework for Runtime Power Introspection in High-Volume Commercial Microprocessors. In *MICRO*.
[78] C. Xu et al. 2022. SNS's Not a Synthesizer: A Deep-Learning-Based Synthesis Predictor. In *ISCA*.
[79] K. Yu et al. 2006. Active Learning via Transductive Experimental Design. In *ICML*.
[80] L. Yu et al. 2017. Seqgan: Sequence generative adversarial nets with policy gradient. In *AAAI conference on artificial intelligence*.
[81] Y. Zhang et al. 2020. GRANNITE: Graph neural network inference for transferable power estimation. In *DAC*.
[82] J. Zhao et al. 2017. COMBA: A comprehensive model-based analysis framework for high level synthesis of real applications. In *ICCAD*.
[83] J. Zhao et al. 2019. Machine learning based routing congestion prediction in FPGA high-level synthesis. In *DATE*.
[84] Z. Zhao et al. 2017. Source-Level Performance, Energy, Reliability, Power and Thermal (PERPT) Simulation. *IEEE TCAD* 36, 2 (2017), 299–312.
[85] X. Zheng et al. 2017. LACross: Learning-Based Analytical Cross-Platform Performance and Power Prediction. *IJPP* 45, 6 (2017), 1488–1514.
[86] X. Zheng et al. 2017. Sampling-Based Binary-Level Cross-Platform Performance Estimation. In *DATE*.
[87] Y. Zhou et al. 2019. PRIMAL: Power inference using machine learning. In *DAC*.