2022 IEEE 40th International Conference on Computer Design (ICCD) | 978-1-6654-6186-3/22/$31.00 ©2022 IEEE | DOI: 10.1109/ICCD56317.2022.00039

AI-assisted Synthesis in Next Generation EDA: Promises, Challenges, and Prospects

Nan Wu[1], Yuan Xie[2], Cong Hao[3]
[1]University of California - Santa Barbara, [2]Alibaba DAMO Academy, [3]Georgia Institute of Technology
nanwu@ucsb.edu, yuanxie@gmail.com, callie.hao@gatech.edu

***Abstract*—Despite the great advance achieved by electronic design automation (EDA) tools, there is still a long way towards hardware agile development, whose ultimate goal is to reduce chip development cycles from years to months or even weeks. Hardware development typically involves many optimization-evaluation iterations, indicating that (1) fast and accurate quality-of-result (QoR) evaluation and (2) efficient optimization, either independently or integrally, will conspicuously improve the development efficiency. Specifically, targeting high-level synthesis and logic synthesis, we investigate (1) the power of exploiting graph neural networks (GNNs) for generalizable and accurate performance predictions, (2) the efficacy of applying reinforcement learning (RL) for design exploration, and (3) the superiority of combining GNN and RL to solve EDA problems. Experimental results demonstrate the promises of infusing intelligence into design synthesis and EDA tools. On top of current endeavors, we summarize the challenges in the respective EDA contexts and the prospects toward next generation EDA tools.**

***Index Terms*—machine learning for EDA, high-level synthesis, logic synthesis, graph neural network, reinforcement learning, performance modeling, design space exploration**

I. Introduction

Moore's law [1], [2] has been powering the integrated circuit revolutions since 1960s, which doubles the transistor density every 18 months. Even if the target cadence of Moore's law is slipping [3], the electronic industry continues to move to larger-scale, more complex, and heterogeneous designs and systems, to keep pace with the exponentially growing compute demand of different applications [4]. However, with the increasing complexity in hardware designs, from the time-to-market aspect, near 70% of application-specific integrated circuit (ASIC) or field-programmable gate array (FPGA) projects are completed behind schedule in 2020 [5]; from the cost aspect, the development costs of leading-edge electronic designs are skyrocketing [6]; from the tool aspect, existing electronic design automation (EDA) tools cannot adequately address emerging hardware development [7]. These all herald the necessity of hardware agile development, with the ultimate goal to reduce chip development cycles from years to months or even weeks. One example is the Intelligent Design of Electronic Assets (IDEA) program [8], aiming to accelerate development cycle of next-generation electronic systems with reduced labors, costs, and design complexity barriers.

Hardware development is an iterative process involving many optimization-evaluation iterations. Fig. 1 depicts the typical design flow from behavioral programs to circuit designs (e.g., layout/bitstream), including high-level synthesis (HLS), logic synthesis, and physical implementation. Notably, every design stage associates with **evaluation phases** that should accurately assess the quality of results (QoRs) of circuit designs to guide design explorations and **optimization phases** that should sufficiently explore design knobs to meet specified performance requirements. Traditional EDA tools usually provide either accurate yet time-consuming or fast yet inaccurate QoR estimations [9]–[12], and extensive manual efforts are required for design space exploration (DSE) to satisfy diverse performance, resource, and power targets. These all result in long time-to-market, which is further exacerbated by the explosion of modern hardware system complexity and technology scaling. Given the avidity toward hardware agile development and productivity boost, it is highly expected to infuse more intelligence into EDA tools to enable fast and accurate QoR evaluation and efficient optimization, either independently or integrally, so that design iterations are conspicuously sped up to improve development efficiency.

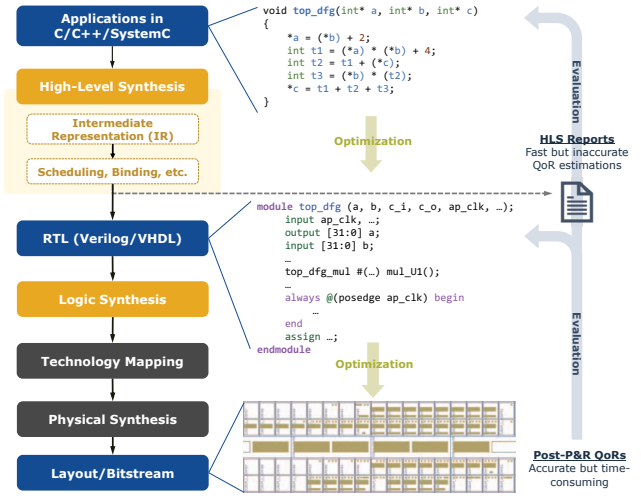

Fig. 1. Development flow from behavioral programs in high-level languages to circuit designs.

Recent years have witnessed the emergence of machine learning (ML) applied for computer architecture and systems [13], revealing the great potentials of ML-based performance modeling and ML-assisted design optimization. In this paper, we investigate **how ML techniques can be embraced**

2576-6996/22/$31.00 ©2022 IEEE
DOI 10.1109/ICCD56317.2022.00039

in HLS and logic synthesis for ① fast and accurate QoR evaluation and ② efficient optimization.

- In Section II and Section III, we explore the power of various graph neural networks (GNNs) for fast, accurate, and generalizable performance predictions in HLS and logic synthesis [12], [14], since data are naturally represented in graphs for many EDA tasks.
- In Section IV, we study the efficacy of flexible and automatic design exploration in HLS enabled by reinforcement learning (RL), whose superiority is further enhanced by the assistance of GNNs to transfer knowledge and past experiences to new or unseen applications [11], [15].
- In Section V, we discuss the challenges and prospects on top of current endeavors, regarding data collection, model selection, and deployment scenarios. We envision ML-based techniques could be the impetus to next-generation EDA and hardware agile development.

II. HLS Performance Predictions with GNNs

HLS expedites ASIC/FPGA development by automatic transformation from behavioral descriptions in high-level languages (C/C++, etc.) to functionally equivalent RTL designs with various resource/performance trade-offs.

A. Why, How, and When to Predict

There are three fundamental questions for HLS performance predictions: ***why***, ***how***, and ***when***.

Why to predict? Though HLS tools can greatly speed up circuit design, they still require minutes to hours for design synthesis before place and route, and the reported hardware performance metrics are far from accurate [9]–[12]. This prevents designers from sufficient design exploration and optimization. Thus, a quick and accurate QoR evaluation at early design stages, even before HLS, is highly expected.

How to predict? For HLS-based hardware designs, post-implementation metrics (e.g, resource usage and timing) are of high interest. Classic approaches generally use analytical models [16]–[18], which are more suitable for well structured data flows. Existing ML-based approaches attempt linear regression, ANN, support vector machine, random forest, and ensemble models [9], [19]–[21], promising but requiring heavy feature engineering after HLS execution. One major concern of these approaches is the limited generalization capability, which have to re-run HLS or implementation for each new and unseen design to collect features. In this regard, more advanced methods are expected for generalizable and accurate HLS performance predictions.

When to predict? HLS performance predictions can be conducted at different synthesis stages, based on the features employed in the ML-based predictors. There are three major sources for feature extraction: HLS directives [22]–[24], intermediate representations (IRs) after HLS front-end compilation [10], [25], [26], and HLS synthesis reports [9], [20], [27]. In general, an early and timely prediction benefits agile development, but little domain-specific knowledge is exposed at this stage, which probably hurts prediction accuracy. Thus, there awaits a comprehensive comparison among prediction strategies at different HLS stages in terms of prediction accuracy and timeliness.

B. Three Prediction Strategies using GNNs

In response to existing limitations (i.e., weak generalizability, unreconciled accuracy and timeliness), we propose three GNN-based prediction approaches [12] with various trade-offs between timeliness and accuracy (as shown in Fig. 2). To make it *timely*, we perform predictions based on the IR graph of a program, i.e., data flow graph (DFG) and control data flow graph (CDFG), since these IR graphs can be quickly extracted after the front-end compilation [28] within seconds. To make it *generalizable*, we exploit the inductiveness of GNNs to make accurate predictions for completely unseen designs without retraining. Specifically, the three approaches are:

- ***Off-the-shelf approach***. The first approach directly predicts post-implementation performance metrics based on IR graphs. The features are extracted right after HLS front-end compilation, leading to the earliest predictions but with compromised accuracy due to the ignorance of hardware-specific information.
- ***Knowledge-rich approach***. The second approach draws support from domain information distilled from intermediate HLS results (i.e., partial execution of HLS but no implementation): the resource usage associated with each node in IR graphs. Armed with rich domain knowledge, this approach emphasizes more on prediction accuracy, especially for resource estimation, yet compromises timeliness and efficiency since HLS tools do take some time to generate intermediate results.
- ***Knowledge-infused approach***. The third approach is a *hierarchical GNN-based* prediction strategy that reaps advantages of the previous two approaches: not only does it make the earliest prediction but also benefits from domain knowledge with almost zero overhead during inference. The knowledge infusion is achieved by decoupling the prediction task into two steps: the first step is node-level classification for resource types, in which the domain knowledge is infused during the training phase; the second step is graph-level regression that estimates the numerical resource usage and timing on top of the self-inferred domain knowledge.

C. Promises in Accuracy, Timeliness, and Generalizability

Predicting post-implementation resource/timing from IR graphs is using GNNs to approximate the set of sophisticated heuristics and mapping rules used by HLS scheduling/binding and logic/physical synthesis during the design flow. We launch discussions from three aspects: ① how different applications (i.e., graph structures) influence prediction accuracy; ② which properties of existing GNN models would help improve accuracy; ③ what domain-specific insights can be derived to facilitate future graph representation learning on fast and generalizable QoR evaluation.

Different graph structures. In the off-the-shelf approach, we screen 14 state-of-the-art GNN models [12]. Table I

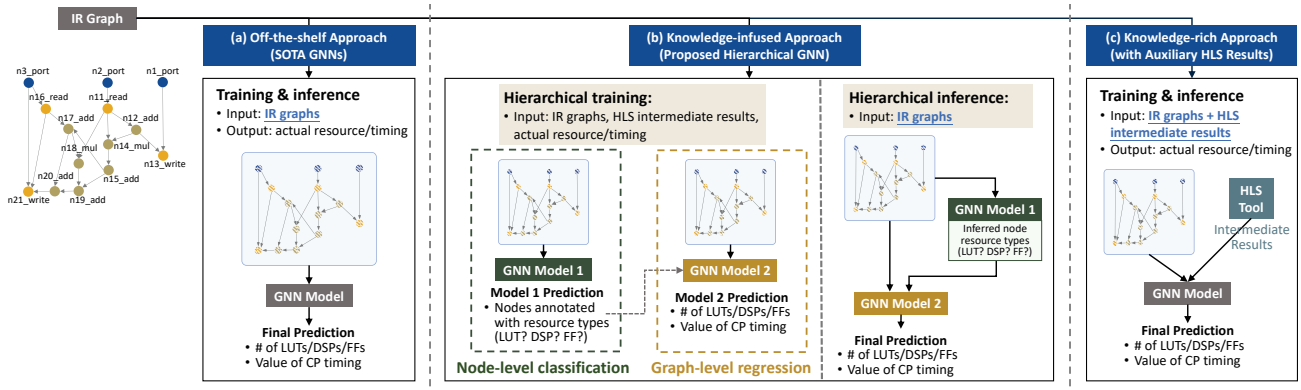

Fig. 2. Our three proposed approaches: (a) off-the-shelf approach, which makes predictions at the earliest stage based on IR graphs; (b) knowledge-infused approach, which breaks the prediction task into two steps, node-level resource type classification and graph-level resource usage and timing regression, striving a balance between timeliness (i.e., making predictions at the earliest stage with IR graphs) and accuracy (i.e., using self-inferred domain-specific information); (c) knowledge-rich approach, which needs to obtain auxiliary information after partial execution of HLS, producing accurate but relatively late predictions.

exhibits the mean absolute percentage error (MAPE) of predictions on resource usage (DSP/LUT/FF) and critical path timing (CP) of synthetic programs. In general, CDFGs have larger MAPE, since message-passing-based GNN models have limited expressiveness on graphs with many loops [29], and the additional nodes/edges representing control states and dependency introduced by control signals easily confuse GNN models during resource prediction.

GNN model analysis. PNA and RGCN generally show superior performance, implying two takeaways: ① the relational information (i.e., edge information) is important in IR graphs, since it represents data or control dependency, or a mix of both, which is a critical basis in logic synthesis and impacts resource allocation; ② equipped with multiple aggregators, PNA is more powerful to characterize different neighborhood information, thus providing better prediction accuracy.

Domain-specific insights. ① Resource. The key to making precise DSP prediction is to distinguish major computation nodes most likely to use DSPs. Similarly, effective extraction of memory-related nodes will greatly benefit FF predictions. As LUTs are involved in the entire graph (as compute units and glue logic to circuit components), graph-level understanding is important. ② Timing. Compared with resource predictions, CP predictions show relatively lower MAPE and better consistency between DFGs and CDFGs, which is probably because timing is local information and insensitive to graph sizes.

Accurate, timely, and generalizable. Intuitively, the more domain information is leveraged, the more accurate predictions are provided, whereas the longer time would be taken for feature collection. The knowledge-infused approach strikes a great balance between accuracy and timeliness. As shown in Fig. 2(b), with hierarchical training both the node-level and the graph-level GNN models are approximating simplified design heuristics: the node-level classification aims to understand the preference of resource types on different nodes; the graph-level regression focuses on globally estimating resource sharing and interference among nodes. With hierarchical inference, the domain knowledge infused during training can be self-inferred when encountering unseen designs, leading to improved prediction accuracy from the earliest design stage.

TABLE I
TESTING MAPE WITH DIFFERENT GNN MODELS ON SYNTHETIC PROGRAMS. THE TOP TWO PERFORMANT MODELS ARE MARKED IN BOLD.

	DFG				CDFG			
	DSP	LUT	FF	CP	DSP	LUT	FF	CP
GCN	16.31%	16.49%	21.27%	**6.12%**	25.30%	28.64%	38.34%	8.79%
GCN-V	15.72%	15.93%	21.64%	6.36%	17.31%	33.93%	39.94%	**8.13%**
SGC	42.12%	23.93%	30.61%	7.92%	44.01%	60.87%	53.50%	10.32%
SAGE	15.18%	14.01%	17.11%	**6.12%**	17.01%	28.09%	39.11%	**8.25%**
ARMA	19.12%	13.46%	16.87%	6.50%	18.47%	**25.21%**	32.15%	8.42%
PAN	15.24%	14.13%	17.23%	6.38%	16.88%	32.65%	44.36%	8.54%
GIN	15.52%	16.10%	22.08%	6.58%	15.47%	28.48%	38.82%	8.76%
GIN-V	15.04%	16.17%	23.09%	6.40%	17.94%	29.40%	48.64%	8.59%
PNA	**12.65%**	**11.64%**	**14.41%**	6.26%	**14.71%**	**22.86%**	**26.47%**	8.87%
GAT	26.22%	22.64%	27.74%	8.30%	28.66%	46.19%	54.73%	10.32%
GGNN	15.40%	13.64%	16.94%	6.47%	16.28%	28.05%	31.88%	8.50%
RGCN	**13.27%**	13.03%	**15.09%**	6.14%	**15.03%**	26.33%	**25.52%**	8.72%
UNet	18.40%	14.90%	19.17%	6.61%	18.92%	32.83%	53.06%	9.02%
FiLM	20.05%	**12.50%**	16.94%	6.27%	17.42%	26.97%	27.35%	8.67%

TABLE II
TESTING MAPE ON REALISTIC APPLICATIONS. -I IS KNOWLEDGE-INFUSED APPROACH; -R IS KNOWLEDGE-RICH APPROACH.

	HLS	RGCN	RGCN-I	RGCN-R	PNA	PNA-I	PNA-R
DSP	26.07%	45.61%	40.89%	32.90%	40.06%	21.95%	**15.20%**
LUT	871.56%	66.23%	30.91%	24.08%	56.34%	21.45%	**16.96%**
FF	322.86%	101.20%	38.75%	27.72%	47.65%	20.10%	**17.42%**
CP	32.09%	8.13%	5.35%	5.83%	8.68%	4.80%	**3.97%**

Table II shows MAPE of the three proposed approaches and Vitis HLS [30] on real-case applications [31]–[33]. Compared with Vitis HLS, PNA-based knowledge-infused approach (denoted as PNA-I) and knowledge-rich approach (denoted as PNA-R) reduce prediction errors by 1.2× to 40.6× and 1.7× to 51.4×, respectively. Such results empirically demonstrate ① generalization capability not only from seen to unseen designs but also from synthetic to realistic applications, and ② accuracy and timeliness conspicuously surpassing HLS tools.

III. LOGIC SYNTHESIS QOR PREDICTIONS WITH MULTI-MODAL GRAPH LEARNING

Logic synthesis converts RTL designs into optimized gate-level representations. The goal is to reduce the amount of

required hardware or the critical path delay by sequences of logic transformations, referred to as logic synthesis flows.

A. Challenges in ML-assisted Logic Optimization

Recently, ML-based approaches are employed to predict QoRs of logic synthesis flows [34], [35], but there remain unresolved challenges and requirements.

- There is no one-for-all solution. Commercial EDA tools usually provide reference synthesis flows [36] developed by experts, but such flows do not uniformly perform well. This suggests the importance of **design-specific** synthesis flows.
- The **transformation order** in synthesis flows should be well captured, which majorly determines final QoRs.
- Existing approaches do not generalize across designs nor flow lengths [34], [35]. Aiming at a practical use of ML-based performance modeling, the **generalization** across different designs and flow lengths is a necessity.

B. Hybrid Graph Models using Spatio-Temporal Information

To address the aforementioned issues, we present a fast, accurate, and generalizable ML approach for QoR estimations of logic synthesis flows, exploiting spatio-temporal information, namely LOSTIN [14]. Two models are explored: ① a GNN for spatial information learning, armed with a supernode to encode temporal information (denoted as GNN-S); ② a hybrid model, composed of a GNN for spatial learning and an LSTM for temporal learning (denoted as GNN-H).

GNN-S: GNN with supernode. Inspired by the idea that introducing a supernode to graphs can collect and redistribute global information [37], we propose to leverage a supernode to represent synthesis flows. As the supernode is virtually connected to all the nodes in the original graph, temporal information is directly injected into the circuit graph (Fig. 3(a)).

GNN-H: GNN with LSTM. Since synthesis flows are naturally represented in sequences, an alternative is to leverage a sequence processing model to distill the temporal information. The specific model employed is LSTM, which excels at handling order dependence and variable-length flows. As shown in Fig. 3(b), we separately generate a sequence embedding for synthesis flow representation and a graph embedding for circuit representation, which are then concatenated for downstream predictions. This scheme not only significantly reduces the training complexity and memory overheads but is more efficient to fuse each source of input information.

C. Promises with Multi-Modal Graph Learning

We select circuit designs from the EPFL benchmark [38]. The logic synthesis flows are generated by the logic synthesis tool ABC [39]. Table III shows MAPE of QoR predictions on designs unseen during training. ① The LSTM-based model [35] suffers from a large accuracy degradation for unseen designs, indicating limited generalization capability. ② GNN-S slightly outperforms the LSTM-based model by 3% and 9% in area and delay prediction, respectively. ③ GNN-H maintains its high prediction accuracy, demonstrating extraordinary generalization capability.

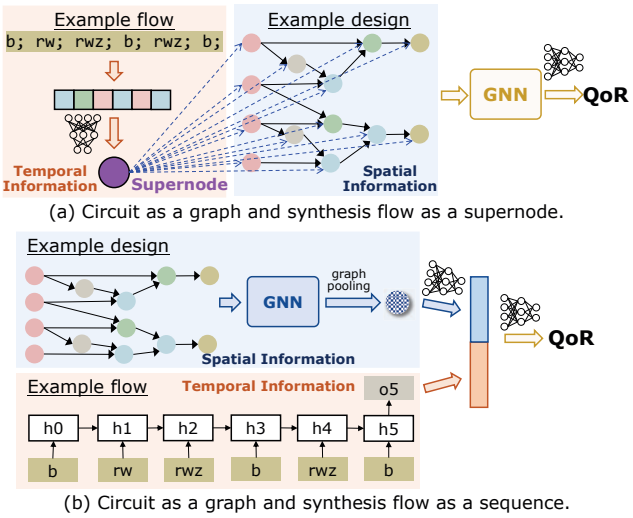

Fig. 3. The proposed approach to predicting QoR after applying logic synthesis flows on hardware designs. (a) GNN-S: the proposed GNN with supernode. (b) GNN-H: the proposed hybrid GNN with LSTM.

TABLE III
COMPARISON WITH LSTM [35] IN THE **INDUCTIVE** SCENARIO.

	Area (MAPE)			Delay (MAPE)		
	LSTM	GNN-S	GNN-H	LSTM	GNN-S	GNN-H
multiplier	57.82%	9.39%	2.45%	38.21%	17.89%	1.75%
sin	66.09%	64.48%	2.34%	45.94%	54.44%	2.32%
sqrt	29.03%	39.25%	4.83%	38.03%	15.75%	2.09%
square	38.59%	13.96%	2.86%	47.52%	31.34%	2.41%
voter	27.38%	76.49%	3.08%	42.19%	46.54%	0.96%
MEAN	43.78%	40.71%	3.11%	42.38%	33.20%	1.91%

GNN-S vs. GNN-H. We compare GNN-S and GNN-H regarding temporal information characterization. In GNN-S, first, the supernode embedding is insensitive to the order of logic transformations; second, with message passing, the original temporal information in the supernode is gradually faded in other nodes; third, simply adding a supernode to original graphs may not be an efficient approach to fusing information from different modalities. By contrast, GNN-H takes advantages of both GNN and LSTM to extract spatio-temporal information in a decoupled manner: the LSTM directly characterizes temporal information from synthesis flows; the GNN focuses on representing spatial structures of circuit designs. These separately learned embeddings have better expressiveness for each source of input information, thus providing a better foundation for downstream tasks.

Multi-modal graph representation learning. Graph representation learning has evolved from single-modal to multi-modal [40], which inspires the adoption of multi-model graph learning for circuit quality evaluation, since the final QoR of circuit designs is dependent on both circuit structures and logic synthesis flows. Our investigation with GNN-S and GNN-H shows that efficient approaches to extracting features and fusing information from different modalities can conspicuously improve representation power. Multi-modal graph representation learning, which integrates the knowledge from other

learning schemes with the conventional graph representation learning, is expected to provide more versatility for EDA tasks.

Generalization to other transformations. Though the main focus is the generalization across different circuit designs, a more practical case as synthesis tools usually hold a fixed set of transformations to be applied [36], [41], we briefly discuss the feasibility of generalizing to additional transformations. First, one of the preprocessing steps for LSTM-based models, i.e., the tokenization of logic transformations, includes a special token designed for transformations that are unknown during training yet met in testing. Second, some out-of-vocabulary techniques [42] can be adopted to improve the generalization capability to new transformations.

IV. Flexible, Fine-grained, and Efficient DSE in HLS using GNN and RL

Despite the great success of HLS tools, we observe several unresolved challenges: ① hard-to-predict quality of the generated RTL designs, ② concealed optimization opportunities due to the high-level abstractions, and ③ inflexible or non-automatic design exploration among different objectives. The GNN-based performance predictors (GPP) discussed in Section II well address the first challenge. In this section, we introduce how the rest challenges are solved by RL.

A. Challenges in HLS Design Exploration

Concealed optimization opportunities. The high-level abstraction in HLS conceals further optimization opportunities. While guidelines of HLS code optimization toward different design objectives are well investigated [43], they often focus on coarse-grained optimization in the loop/array/function-level and manual efforts for fine-grained exploration (such as in the operator-level) are still required. Motivated by the necessity of fine-grained DSE, we propose a **code transformer (CT)**, which breaks up the high-level abstractions by exposing operations in behavioral descriptions.

Inflexible DSE among different objectives. Traditional HLS DSE uses meta-heuristics, such as genetic algorithms (GA) [44], simulated annealing (SA) [45], particle swarm optimization (PSO) [46], and ant colony optimization (ACO) [47]. For ML-based DSE, active learning [23] and Bayesian optimization [48], [49] are popular options. However, there are several limitations. ① Meta-heuristics require explorations from scratch for every new design and do not benefit from previous experiences, leading to long search time and degraded solution quality. ② Many DSE approaches need to invoke HLS and implementation process to validate newly generated solutions during optimization, which is time-consuming. ③ Not all DSE approaches are suitable for large design spaces. ④ Current DSE usually sacrifices design latency for less resource or vise versa [50], leaving flexible trade-offs among other objectives unexplored. One potential alternative is to trade one type of resource for another (e.g., `LUT` and `DSP` in FPGA), which only can be done through tedious manual efforts. Motivated by the necessity and difficulty of efficient and flexible DSE, we propose an **RL**-based **M**ulti-objective **D**SE **(RLMD)** engine for optimal resource allocation strategies under user-specified constraints, which can also provide Pareto solutions between different objectives.

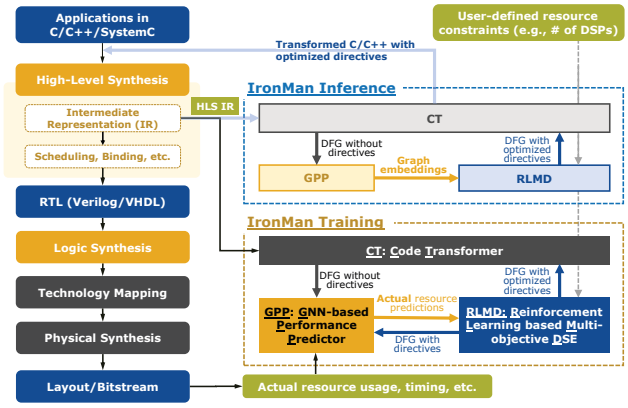

Fig. 4. IronMan is a learning-based framework composed of CT, GPP, and RLMD. During training, IronMan takes HLS C/C++ code and IRs as inputs and the actual RTL performance (e.g., resource and timing) as the ground truth to train GPP and RLMD. Specifically, GPP is trained to predict `LUT`/`DSP`/`CP` based on DFGs and applied directives as node features; RLMD concatenates the graph embeddings generated by GPP with the metadata of input DFGs to compose state representations, outputs a binary probability distribution of whether to apply resource pragmas, and receives predicted `LUT`/`DSP`/`CP` from GPP to derive rewards. During inference, the well-trained GPP provides graph embeddings and performance predictions to RLMD; the trained RLMD either finds optimized directives that satisfy user-specified design constraints such as available resources, or generates Pareto-solutions with various trade-offs between different resource types.

B. IronMan: GNN + RL for DSE

As shown in Fig. 4, we integrate CT, GPP, and RLMD into a framework, **IronMan** [11], [15]. **CT** is the interface between GPP/RLMD and HLS tools, which extracts DFGs after HLS front-end compilation to release more optimization opportunities and then re-generates synthesizable C/C++ code based on the optimized DFGs. **GPP** is a highly accurate GNN-based performance predictor as introduced in Section II.

RLMD is an RL-based DSE engine. As a case study of IronMan, the specific problem solved is to find a resource allocation solution that strictly meets DSP constraints, or to find Pareto solutions between `DSP`s and `LUT`s (or `CP` timing) on FPGAs, without sacrificing the compute latency. Thus, RLMD observes the raw DFG, its graph embedding, and user-specified constraints as states, and takes actions to decide whether to use `LUT`s to implement a multiplication node by assigning resource pragmas. The reward function is defined as a negative weighted sum of multiple metrics, such as the predicted `LUT` usage, the predicted CP timing, and the discrepancy between predicted DSP usage and target DSP usage, which automatically enables multi-objective optimization by adjusting the weights of different metrics. The optimization goal is to maximize the average expected rewards over all training DFGs, such that RLMD can figure out various trade-offs across various DFGs and DSP constraints. RLMD is equipped with two different RL methods, actor-critic (AC) and

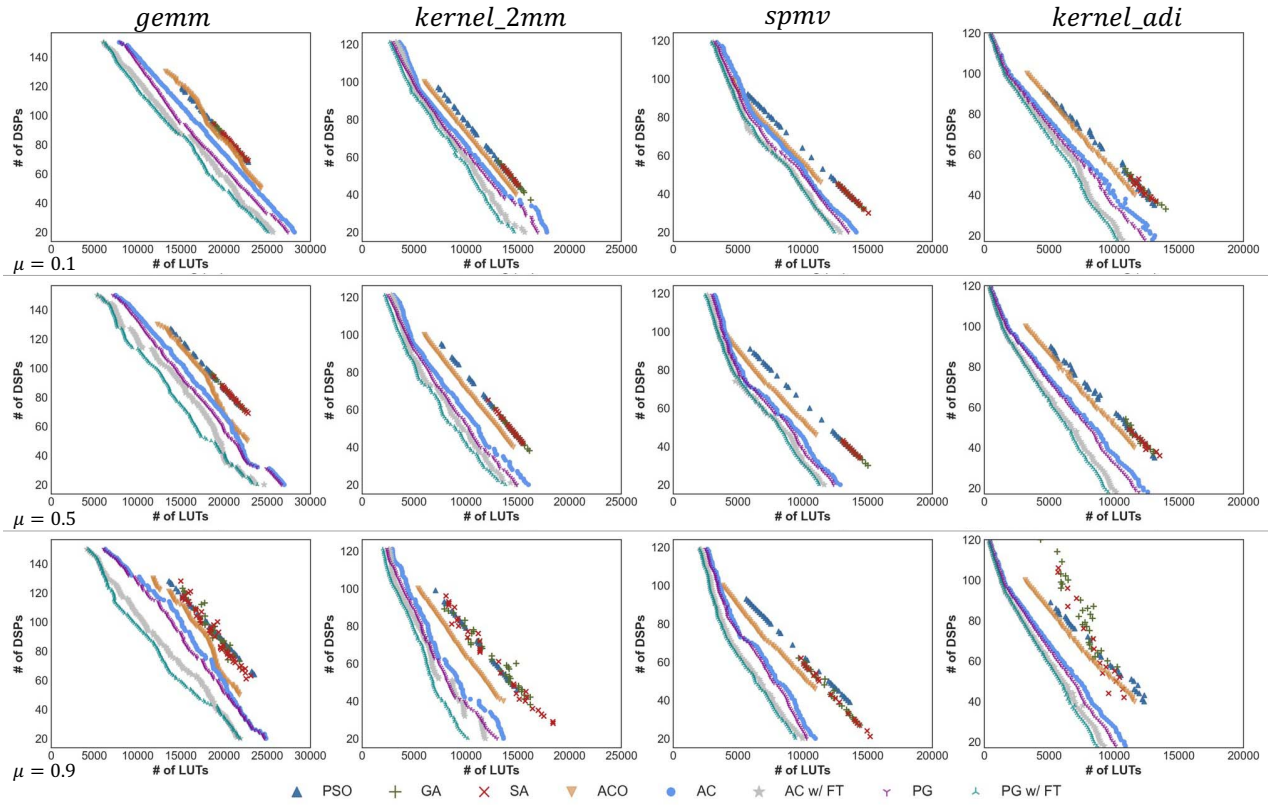

Fig. 5. Pareto solutions found by RLMD, PSO, GA, SA, and ACO on four real-case benchmarks, *gemm*, *kernel_2mm*, *spmv*, and *kernel_adi*, with unchanged latency (i.e., the number of clock cycles of the synthesized design). The toolbox of RLMD involves AC, PG, either with or without a fine-tuning (FT) step. Different settings of μ indicate that different importance is assigned to LUT utilization and CP timing during the optimization.

policy gradient (PG) [51], providing the flexibility to choose a more proper optimization scheme for different cases.

RLMD fine-tuning. Given a new DFG, the simplest way is to directly apply the pre-trained RLMD for inference. When higher quality solutions are desired, the pre-trained RLMD can be further fine-tuned on a particular DFG. The fine-tuning step provides the flexibility to balance between a quick solution with the pre-trained RLMD (which has learned rich knowledge of resource allocation strategies on other DFGs) and a longer yet better one for a particular DFG.

C. Promises in Flexible, Fine-grained, and Efficient DSE

We demonstrate the end-to-end benefits on benchmarks from real-world applications in Fig. 5. Obviously, RLMD, either with AC or PG method, outperforms GA, SA, PSO, and ACO by a large margin. In terms of multi-objective optimization, given DSP usage constraints, the solutions found with $\mu = 0.9$ often consume fewer LUTs, compared with those found with $\mu = 0.1$. This indicates that RLMD can properly balance between LUT usage and CP timing when different importance is assigned to different metrics, whereas the heuristic-based methods cannot explicitly leverage the trade-offs among multiple objectives.

These promising results show great potentials of applying RL for DSE in HLS. We briefly explain the reasons for the superiority of IRONMAN: ① the design space grows exponentially with the size of DFGs, different graph topologies, and various data precisions; RL agents can explore design space proactively and learn from past experiences; after training, it can generalize to new problems with minimal fine-tuning efforts, revealing better scalability and efficacy. ② by carefully defining reward functions, RL agents can achieve multi-objective optimization automatically, eliminating manual efforts to craft useful heuristics. ③ with the help of CT, RLMD can conduct the fine-grained DSE that are not supported by any of the existing DSE approaches; with the help of GPP, the informative state representations not only significantly benefit the learning process of RLMD but also enable RLMD to better generalize across different DFG topologies.

V. CHALLENGE AND PROSPECT

In this section, we discuss challenges and prospects of exploiting ML techniques for EDA problems, which span data, algorithms/models, deployment, and long-term targets.

A. Data Collection

Data scarcity. In some EDA problems, such as place and route in physical synthesis, the simulation is extremely expensive.

As ML models usually require enough data to learn underlying statistics and make decisions, this gap between small data and big data often limits the capability of ML-based techniques. From the algorithm side, algorithms that can work with small data await to be developed. From the data side, generative methods can be used to generate synthetic data [52].

Non-perfect data. Even if some EDA tools produce a lot of data (such as simulation-based testing), they are not always properly labeled nor presented in the form suitable to ML models. Thus, possible alternatives are unsupervised learning, semi-supervised learning [53], self-supervised learning [54], or to combine supervised with unsupervised techniques [55].

Generalization to out-of-distribution data. Though synthetic data can help with mitigating the data gap, it is noteworthy that data distribution varies between synthetic and real-case data [56], which often causes data drift or concept drift [57]. This appeals for incorporating out-of-distribution methods [58].

B. Model/Algorithm Development

Multi-level abstraction and optimization. Classical EDA methods usually adopt a bottom-up or top-down procedure, encouraging ML-based techniques to distill hierarchical structures of hardware designs. Potential methods toward multi-level design abstraction and optimization are ① hierarchical RL [59] that has flexible goal specifications and can learn goal-directed behaviors in complex environments with sparse feedback and ② multi-agent RL [60] where agents can be fully cooperative, fully competitive, or a mix of the two, enabling versatility of system optimization.

Interpretability. The absence of interpretation regarding model behaviors and decisions limits wider adoption of ML for EDA tasks, since these explanations are important to identify and expose potential problems during training and ensure fidelity of models/algorithms. Thus, efforts in interpretable ML [61], [62] are highly expected to promote production-ready applications of ML for EDA.

C. Implementation and Deployment Improvement

Online vs. offline. When deploying ML-based techniques for EDA tasks, it is crucial to deliberate design constraints under different scenarios. ① ML-based techniques are deployed online or during runtime, no matter the training phase is online or offline. Obviously, the model complexity and runtime overheads are strictly limited by specific constraints. If online learning is further desired, the design constraint will be more stringent. One promising approach is to employ semi-online learning models, which have been applied to solve some classical combinatorial optimization problems, such as bipartite matching [63] and caching [64]. These models enable smooth interpolation between the best possible online and offline training algorithms. ② ML-based techniques are applied offline to guide hardware design, and once the designing phase is completed, ML models will not be invoked again. Thus, the offline applications can tolerate relatively higher overheads.

Model maintenance. In the case of offline training and online deployment, ML models require regular maintenance and updating to meet performance expectations. ① ML models can be retrained either at a regular interval or when key performance indicators are below certain thresholds. Retraining models regularly, regardless of their performance, is a more direct way, but it requires a clear understanding of how frequently a model should be updated under its own scenario. The model performance will decline if retraining intervals are too spaced out in the interim. Monitoring key performance indicators relies on a comprehensive panel of measurements that explicitly demonstrate model drift, whereas this may introduce additional hardware/software overhead and incorrect selection of measurements often defeats the intention of this method. ② During the retraining of ML models, there is often a trade-off between newly collected data and previous data. Properly assigning importance of input data would improve retraining efficacy [65].

D. General, Portable, and Agile Hardware Development

Infusing more intelligence into EDA will make great strides toward the landing of hardware agile development.

General. We envisage an ML-based system-wise and holistic framework with a panoramic vision: it should be able to leverage information from different levels of hardware designs in synergy, to thoroughly characterize the behaviors as well as their intrinsically hierarchical abstractions; it should also be able to make decisions in different granularity, to control and improve the hardware precisely and comprehensively.

Portable. The well-designed interfaces between EDA tools and ML-based techniques are expected to facilitate the portability across different platforms, since ML models can perform well without explicit descriptions of the target domain.

Agile. The proliferation of ML-based techniques has more or less transformed the EDA workflow. We expect GNNs make better use of naturally graphical data in the EDA field; we expect deep RL be a powerful and general-purpose tool for many EDA optimization problems, especially when the exact heuristic or objective is obscure; we expect more intelligence will be infused into next-generation EDA tools, to enhance designers' productivity and to thrive in the community.

VI. Conclusion

In this paper, we target HLS and logic synthesis, and discuss ① the power of GNNs for fast, accurate, and generalizable QoR predictions, and ② the efficacy of RL-enabled flexible and automatic design exploration. Standing on current endeavors, we provide a future vision of challenges and prospects of infusing more intelligence for next-generation EDA.

References

[1] G. E. Moore, "Cramming more components onto integrated circuits," *Proc. IEEE*, 1998.

[2] C. A. Mack, "Fifty years of moore's law," *IEEE Trans. Semicond. Manuf.*, 2011.

[3] M. M. Waldrop, "The chips are down for moore's law," *Nature News*, 2016.

[4] OpenAI. (Accessed: 2022-08) Al and compute. [Online]. Available: https://openai.com/blog/ai-and-compute/

[5] H. Foster. (Accessed: 2022-08) The 2020 wilson research group functional verification study. [Online]. Available: https://blogs.sw.siemens.com/verificationhorizons/2020/10/27/prologue-the-2020-wilson-research-group-functional-verification-study/

[6] F. Schirrmeister *et al.* (Accessed: 2022-08) Next generation verification for the era of ai/ml and 5g. Design and Verification Conference and Exhibition, US (DVCon), 2020. [Online]. Available: https://dvcon-proceedings.org/document/next-generation-verification-for-the-era-of-ai-ml-and-5g/

[7] M. Rosker. (Accessed: 2022-08) Evolving the electronics resurgence initiative (eri 2.0). [Online]. Available: https://www.ndia.org/-/media/sites/ndia/divisions/electronics/eri2_ndia_20210421_releaseapproved_34584.ashx

[8] J. Wilson. (Accessed: 2022-08) Intelligent design of electronic assets (idea). [Online]. Available: https://www.darpa.mil/program/intelligent-design-of-electronic-assets

[9] S. Dai *et al.*, "Fast and accurate estimation of quality of results in high-level synthesis with machine learning," in *Proc. FCCM*, 2018.

[10] E. Ustun *et al.*, "Accurate operation delay prediction for fpga hls using graph neural networks," in *Proc. ICCAD*, 2020.

[11] N. Wu *et al.*, "Ironman: Gnn-assisted design space exploration in high-level synthesis via reinforcement learning," in *Proc. GLSVLSI*, 2021.

[12] N. Wu *et al.*, "High-level synthesis performance prediction using gnns: Benchmarking, modeling, and advancing," in *Proc. DAC*, 2022.

[13] N. Wu and Y. Xie, "A survey of machine learning for computer architecture and systems," *ACM Comput. Surveys*, 2022.

[14] N. Wu *et al.*, "Lostin: Logic optimization via spatio-temporal information with hybrid graph models," in *Proc. ASAP*, 2022.

[15] N. Wu *et al.*, "Ironman-pro: Multi-objective design space exploration in hls via reinforcement learning and graph neural network based modeling," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, 2022.

[16] J. Zhao *et al.*, "Comba: A comprehensive model-based analysis framework for high level synthesis of real applications," in *Proc. ICCAD*, 2017.

[17] A. B. Perina *et al.*, "Lina: Timing-constrained high-level synthesis performance estimator for fast dse," in *Proc. ICFPT*, 2019.

[18] J. Zhao *et al.*, "Performance modeling and directives optimization for high-level synthesis on fpga," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, 2019.

[19] K. O'Neal *et al.*, "Hlspredict: Cross platform performance prediction for fpga high-level synthesis," in *Proc. ICCAD*, 2018.

[20] H. M. Makrani *et al.*, "Pyramid: Machine learning framework to estimate the optimal timing and resource usage of a high-level synthesis design," in *Proc. FPL*, 2019.

[21] H. M. Makrani *et al.*, "Xppe: cross-platform performance estimation of hardware accelerators using machine learning," in *Proc. ASP-DAC*, 2019.

[22] H.-Y. Liu and L. P. Carloni, "On learning-based methods for design-space exploration with high-level synthesis," in *Proc. 50th DAC*, 2013.

[23] P. Meng *et al.*, "Adaptive threshold non-pareto elimination: Re-thinking machine learning for system level design space exploration on fpgas," in *Proc. DATE*, 2016.

[24] J. Kwon and L. P. Carloni, "Transfer learning for design-space exploration with high-level synthesis," in *Proc. MLCAD*, 2020.

[25] D. Koeplinger *et al.*, "Automatic generation of efficient accelerators for reconfigurable hardware," in *Proc. ISCA*, 2016.

[26] J. Zhao *et al.*, "Machine learning based routing congestion prediction in fpga high-level synthesis," in *Proc. DATE*, 2019.

[27] Z. Lin *et al.*, "Hl-pow: A learning-based power modeling framework for high-level synthesis," in *Proc. ASP-DAC*, 2020.

[28] A. V. Aho *et al.*, *Compilers: principles, techniques, & tools*. Pearson Education India, 2007.

[29] H. Maron *et al.*, "Provably powerful graph networks," *Proc. NeurIPS*, 2019.

[30] Vitis, *Vitis High-Level Synthesis User Guide (UG1399)*, Accessed: 2022-08, https://docs.xilinx.com/r/en-US/ug1399-vitis-hls.

[31] B. Reagen *et al.*, "Machsuite: Benchmarks for accelerator design and customized architectures," in *Proc. IISWC*, 2014.

[32] Y. Hara *et al.*, "Proposal and quantitative analysis of the chstone benchmark program suite for practical c-based high-level synthesis," *Journal of Information Processing*, 2009.

[33] L.-N. Pouchet and T. Yuki. (2016) Polyhedral benchmark suite. [Online]. Available: http://web.cs.ucla.edu/~pouchet/software/polybench/

[34] C. Yu *et al.*, "Developing synthesis flows without human knowledge," in *Proc. DAC*, 2018.

[35] C. Yu and W. Zhou, "Decision making in synthesis cross technologies using lstms and transfer learning," in *Proc. MLCAD*, 2020.

[36] Synopsys. (Accessed: 2022-08) Lynx design system. [Online]. Available: https://www.synopsys.com/content/dam/synopsys/implementation&signoff/datasheets/lynx-design-system-ds.pdf

[37] J. Gilmer *et al.*, "Neural message passing for quantum chemistry," in *Proc. ICML*, 2017.

[38] L. Amarú *et al.*, "The epfl combinational benchmark suite," in *Proc. 24th Int. Workshop on Logic & Synthesis*, 2015.

[39] R. Brayton and A. Mishchenko, "Abc: an academic industrial-strength verification tool," in *Proc. CAV*, 2010.

[40] A. Holzinger *et al.*, "Towards multi-modal causability with graph neural networks enabling information fusion for explainable ai," *Information Fusion*, 2021.

[41] Cadence, "Genus synthesis solution," Accessed: 2022-08. [Online]. Available: https://www.cadence.com/en_US/home/tools/digital-design-and-signoff/synthesis/genus-synthesis-solution.html

[42] Z. Hu *et al.*, "Few-shot representation learning for out-of-vocabulary words," in *Proc. ACL*, 2019.

[43] J. de Fine Licht *et al.*, "Transformations of high-level synthesis codes for high-performance computing," *IEEE Trans. Parallel Distrib. Syst.*, 2020.

[44] B. C. Schafer, "Parallel high-level synthesis design space exploration for behavioral ips of exact latencies," *ACM TODAES*, 2017.

[45] B. C. Schafer *et al.*, "Adaptive simulated annealer for high level synthesis design space exploration," in *Proc. VLSI-DAT*, 2009.

[46] Y. Zhang *et al.*, "A comprehensive survey on particle swarm optimization algorithm and its applications," *Math. Problems in Eng.*, 2015.

[47] D. Liu and B. C. Schafer, "Efficient and reliable high-level synthesis design space explorer for fpgas," in *Proc. FPL*, 2016.

[48] Q. Sun *et al.*, "Correlated multi-objective multi-fidelity optimization for hls directives design," in *Proc. DATE*, 2021.

[49] A. Mehrabi *et al.*, "Prospector: synthesizing efficient accelerators via statistical learning," in *Proc. DATE*, 2020.

[50] B. C. Schafer and Z. Wang, "High-level synthesis design space exploration: Past, present, and future," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, 2019.

[51] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press, 2018.

[52] Y. Ding *et al.*, "Generative and multi-phase learning for computer systems optimization," in *Proc. ISCA*, 2019.

[53] J. E. Van Engelen and H. H. Hoos, "A survey on semi-supervised learning," *Mach. Learn.*, 2020.

[54] D. Hendrycks *et al.*, "Using self-supervised learning can improve model robustness and uncertainty," *Proc. NeurIPS*, 2019.

[55] M. Alawieh *et al.*, "Efficient hierarchical performance modeling for integrated circuits via bayesian co-learning," in *Proc. DAC*, 2017.

[56] N. Wu *et al.*, "Program-to-circuit: Exploiting gnns for program representation and circuit translation," *arXiv preprint arXiv:2109.06265*, 2021.

[57] A. Tsymbal, "The problem of concept drift: definitions and related work," *Computer Science Department, Trinity College Dublin*, 2004.

[58] H. Li *et al.*, "Ood-gnn: Out-of-distribution generalized graph neural network," *IEEE Trans. Knowl. Data Eng.*, 2022.

[59] T. D. Kulkarni *et al.*, "Hierarchical deep reinforcement learning: integrating temporal abstraction and intrinsic motivation," in *Proc. NeurIPS*, 2016.

[60] K. Zhang *et al.*, "Multi-agent reinforcement learning: A selective overview of theories and algorithms," *Handbook of Reinforcement Learning and Control*, 2021.

[61] L. H. Gilpin *et al.*, "Explaining explanations: An overview of interpretability of machine learning," in *Proc. DSAA*, 2018.

[62] D. V. Carvalho *et al.*, "Machine learning interpretability: A survey on methods and metrics," *Electronics*, 2019.

[63] R. Kumar *et al.*, "Semi-online bipartite matching," in *Proc. 10th Innovations in Theor. Comput. Sci. Conf.*, 2019.

[64] R. Kumar *et al.*, "Interleaved caching with access graphs," in *Proc. 14th ACM-SIAM Symp. on Discrete Algorithms*. SIAM, 2020.

[65] J. Byrd and Z. Lipton, "What is the effect of importance weighting in deep learning?" in *Proc. ICML.*, 2019.