Bulls-Eye: Active Few-Shot Learning Guided Logic Synthesis

Animesh Basak Chowdhury, Benjamin Tan, *Member, IEEE*, Ryan Carey, Tushit Jain, Ramesh Karri, *Fellow, IEEE*, and Siddharth Garg

***Abstract*—Generating suboptimal synthesis transformation sequences ("synthesis recipe") is an important problem in logic synthesis. Manually crafted synthesis recipes have poor quality. State-of-the art machine learning (ML) works to generate synthesis recipes do not scale to large netlists as the models need to be trained from scratch, for which training data is collected using time-consuming synthesis runs. We propose a new approach, Bulls-Eye, that fine-tunes a pretrained model on past synthesis data to accurately predict the quality of a synthesis recipe for an unseen netlist. Our approach achieves 2×–30× runtime improvement and generates synthesis recipes achieving close to 95% quality-of-result (QoR) compared to conventional techniques using actual synthesis runs. We show our QoR beat state-of-the-art approaches on various benchmarks.**

***Index Terms*—Deep learning, graph neural networks, logic synthesis.**

I. Introduction

THERE is a growing interest in using machine learning (ML) techniques in design automation problems [1], [2], [3]. Several problems in electronic design automation (EDA), such as logic synthesis, placement and routing, and VLSI testing are combinatorial optimization problems that require sequential decision making to achieve the target objective. Logic synthesis, the first step in an EDA flow, applies a sequence of synthesis transformations (the "synthesis recipe") to an and-inverter-graph (AIG) representation of a Boolean function to minimize its size or depth. For instance, transformation heuristics in ABC [4], a leading academic synthesis tool, include steps, such as `balance`, `refactor`, `rewrite`, etc. In this work, we focus on ML-guided search for "good" logic synthesis recipes; a problem that is receiving attention.

Traditionally, EDA engineers choose from a set of handcrafted synthesis recipes, such as `resyn2` or `compress2rs` [4] based on intuition and experience. The space of all synthesis recipes is *vast*. Recent work has shown ML-guided exploration of this solution space can yield synthesis recipes that are tailored for each design and outperform handcrafted recipes in terms of quality-of-result (QoR) [5], [6], [7], [8]. One study formulates finding a good synthesis recipe as a classification problem [5] which seeks to distinguish "angel" recipes from "devil" recipes. Another study [6], [7], [8] formulates it as an Markov decision process (MDP) that is solved using reinforcement learning (RL). However, the prior work is deficient on two fronts: 1) scalability to large designs and 2) generalizability to previously unseen netlists. Typically, models are trained **from scratch**, requiring significant training time and a large number of synthesis runs for large unseen designs; and leaving prior experience **untapped**. For this reason, in [12], "benchmarks were excluded from the experimentation which showed significant training times, being large circuits."

ML methods rely on high-quality labeled data (or reward signals for RL). In logic synthesis, the input features are AIGs and synthesis recipes, and the QoRs are labels/rewards. Prior work obtains labels/rewards (i.e., QoRs) by synthesizing the design using a large number of synthesis recipes. This is time-consuming for large designs. An alternative is to use fewer synthesis runs, or even a model trained on previously seen designs, but this yields lower-quality synthesis recipes.

In this article, we propose an ML-guided synthesis solution that addresses these limitations and scales to *large* designs on which existing state-of-the-art (SOTA) solutions fail or time-out. Bulls-Eye has two key ideas: 1) *few-shot learning:* fine-tuning a base model trained on previously seen designs using a few training samples from the new design and 2) *active learning:* fine-tuning is improved by picking synthesis recipes for which we obtain QoRs/labels given a limited synthesis run budget. We illustrate these ideas using the **Bulls-Eye** framework that uses a graph convolutional network (GCN) [9] as a QoR predictor, and simulated annealing (SA) to search the design space. The ideas generalize to other ML-guided synthesis frameworks. Our key contributions are listed as follows.

1) *Bulls-Eye* is an active few-shot learning approach that transfers knowledge from a base model trained on past data to train a new model that predicts QoR of a synthesis run for an unseen netlist. We use SA to find an optimized synthesis recipe for the new netlist.

Manuscript received 31 March 2022; revised 9 August 2022 and 16 October 2022; accepted 12 November 2022. Date of publication 5 December 2022; date of current version 19 July 2023. This work was supported in part by NSF under Award 1553419 and Award 2039607. This article was recommended by Associate Editor S. Gao. *(Corresponding author: Animesh Basak Chowdhury.)*

Animesh Basak Chowdhury, Ramesh Karri, and Siddharth Garg are with the Department of Electrical and Computer Engineering, New York University, Brooklyn, NY 11220 USA (e-mail: abc586@nyu.edu; rkarri@nyu.edu; sg175@nyu.edu).

Benjamin Tan is with the Department of Electrical and Software Engineering, University of Calgary, Calgary, AB T2N 1N4, Canada (e-mail: benjamin.tan1@ucalgary.ca).

Ryan Carey and Tushit Jain are with the ML for SoC Design, Qualcomm Technologies, Inc., San Diego, CA 92121 USA (e-mail: rcarey@qti.qualcomm.com; tushitj@qti.qualcomm.com).

Digital Object Identifier 10.1109/TCAD.2022.3226668

2) *Scalability:* Bulls-Eye's base model is learned on small designs for which training data is easy to generate and transferred using a few, selected synthesis runs (i.e., "shots") to large, unseen netlists. Bulls-Eye produces high-quality synthesis recipes for benchmarks on which SOTA fail or time out.
3) Bulls-Eye achieves 95% of QoR compared to SA using actual synthesis with **3.7**× median runtime speed-up and upto **30**× speed-up on benchmarks having ≥1M nodes compared to SOTA. Bulls-Eye achieves 3%–10% better QoR with an average **1.04**× (versus [8]) and **11.21**× (versus [7]) runtime speedup.

TABLE I
WORK ON ML-BASED SYNTHESIS RECIPE GENERATION

Prior work	Input Features	ML techniques	Max. nodes considered	Model transferability
[10]	MIG	GCN, RL	≤2000	✗
[5]	Flow encoding	CNN	44045	✗
[6]	AIG encoding	RL	176938	†
[11]	-	MAB	30003	✗
[7]	AIG+state encoding	GCN, RL	2675	✗
[8]	AIG+state encoding	GCN,RL	32060	✗
Bulls-Eye	AIG+recipe encoding	GCN,CNN	16216836	✓

† *DRiLLS claims the model can be reused, without empirical evaluation*

II. PRELIMINARIES AND RELATED WORK

A. Logic Synthesis

Logic synthesis transforms a hardware design in a high-level abstraction (e.g., register-transfer level) to a gate-level netlist, ultimately mapping to a user-specified technology library. Typically, logic synthesis tries to minimize the circuit area and meet a specific delay constraint. Logic synthesis is performed in a series of steps: 1) the design is converted to a generic netlist-level representation (e.g., AIG); 2) the netlist undergoes heuristics-based optimization to implement the same functionality using fewer gates and/or reduced depth which correlates to a reduction in the final area and delay of design [2]; 3) technology mapping of the generic netlist using standard cells; and 4) post-mapping optimizations. Consistent with prior works, we focus on the pretechnology mapping optimization of generic gate-level netlists.

ABC [4] is the current baseline framework for optimizing AIG-based netlists. AIGs are directed acyclic graphs (DAGs) and ABC's synthesis transformation heuristics perform local subgraph-level optimization to reduce the AIG structure. ABC's well-known transformations are `refactor`, `rewrite`, `resubstitute` (these reduce the nodes), and balance (for depth-reduction). For $\mathbf{M}$ unique synthesis transformations, $\{\mathbf{T_1}, \mathbf{T_2}, \ldots, \mathbf{T_M}\}$ in a synthesis recipe $\mathbf{S}$, the possible number of synthesis recipes of length $\mathbf{L}$ is $\mathbf{M^L}$ (with repeatable transformation). This indicates that the search space of synthesis recipes is huge. The problem of optimal synthesis recipe generation for an AIG is

$$\underset{\mathbf{S}}{\operatorname{argmin}}\, \mathrm{QoR}(\mathcal{G}(\mathbf{AIG}, \mathbf{S})) \tag{1}$$

where $\mathcal{G}$ is the synthesis function defined as $\mathcal{G} : AIG \times S \longrightarrow AIG$ and QoR is a function that evaluates the *quality* of the synthesis (i.e., lower number of nodes/depth of AIG is better).

B. Prior Work

We summarize prior work on ML for logic synthesis in Table I. Recent enhancements in logic synthesis have built on recent successes of supervised learning and RL approaches for solving sequential decision-making problems [1], [12] in an attempt to find good synthesis recipes in the vast space of possible recipes. Hosny et al. [6], Zhu et al. [7], Peruvemba et al. [8], and Haaswijk et al. [10] generated optimal synthesis by formulating the problem as an MDP and training a policy gradient agent over a collection of synthesis runs on a set of benchmarks. During inference, the trained agent (in the exploit phase) predicts the best synthesis recipe for a given AIG. Hosny et al. [6] and Haaswijk et al. [10] used handcrafted features from AIGs (i.e., number of AND and NOT gates, depth, number of primary inputs and outputs) to represent AIG state. Zhu et al. [7] used handcrafted features with GCN embedding of AIG to represent the current AIG state. These works show results that improve upon handcrafted recipes like `resyn2`. In an alternate approach, Yu et al. [5] classified an unseen synthesis recipe as good ("angel") or bad ("devil") by training a model on the results corresponding to a set of recipes for a design. However, the trained model is design specific; a model trained on one design cannot be reused for another.

III. PITFALLS OF EXISTING APPROACHES

Black-box optimization techniques (e.g., SA or RL methods in prior work) need feedback in terms of the *cost* (or *reward*) of actions; in the logic synthesis context, the action is a synthesis recipe $\mathbf{S}$ and the cost/reward is $\mathcal{G}(\mathbf{AIG}, \mathbf{S})$, obtained by synthesizing **AIG** using $\mathbf{S}$. Typically a large number of cost/reward evaluations, or equivalently, synthesis runs, are required to obtain good solutions. The biggest challenge here is that even a single synthesis run for complex system-on-chip (SoC) designs require several hours or days, limiting the number of synthesis runs given a time budget. Hence, it limits the amount of training data that can be collected and the fraction of design space a black-box optimizer can cover in a limited time. Although RL-based approaches have shown promise in generating good quality synthesis recipes, they cannot scale up to industrial-sized benchmarks [7], [8]. These observations raise three questions for computationally-efficient ML-based logic synthesis.

1) How can we predict the best synthesis recipe for a new, unseen netlist using only a limited or budgeted number of synthesis runs of that netlist?
2) Given a budget, which synthesis recipes should be selected to train an accurate model for the new netlist?
3) How can we efficiently leverage past information, specifically, synthesis runs on previously seen netlists to improve solution quality?

We next present Bulls-Eye, which solves the key roadblocks toward computationally efficient ML-guided logic synthesis.

IV. Bullseye Framework

To reduce the cost of running actual synthesis during black-box optimization, Bulls-Eye trains a QoR predictor, $\hat{\mathbf{F}}(\mathbf{AIG}, \mathbf{S}, \theta)$, which predicts the QoR for a given **AIG** synthesized using an L-length synthesis recipe **S**. We now seek to solve the following problem:

$$\begin{aligned} &\underset{\mathbf{S}}{\operatorname{argmin}}\, \hat{\mathbf{F}}(\mathbf{AIG}, \mathbf{S}, \theta) \\ &\hat{\mathbf{F}} \approx \mathrm{QoR}(\mathcal{G}(\mathbf{AIG}, \mathbf{S})) \end{aligned} \tag{2}$$

where θ represents the QoR predictor's model parameters and ϵ represents model noise that we also seek to minimize.

A. Insights From ML Domain

Unlike ML application in computer vision, the diversity of circuit characterization in terms of size, complexity, and functionalities is not well studied and understood. We borrow two important insights from the core ML community to solve our problem of learning proxy models with limited data.

Insight 1: In computer vision, various deep learning networks learn robust embeddings and are pretrained on large dataset like Imagenet [13]. Pretrained models are fine-tuned for specific applications. We adopt a similar strategy and apply to EDA. Large circuit designs have smaller known IP designs as building blocks. Pretraining a model using labeled data on small-sized IPs followed by fine-tuning can predict important parameters of downstream tasks. Thus, a one-time training with labeled data on small IPs (varied functionality) can be a qualitative QoR predictor.

Insight 2: Few-shot learning [14] learns a hypothesis function with limited labeled data of a target task, given that abundant labeled data is available for different tasks. Effectively, it uses a transfer learning approach for classification/regression tasks using limited labeled data of the target task.

This insight guides us in our problem context for solving the problem of data scarcity, knowledge transfer, and picking the best-possible synthesis scripts (few-shots) to generate the labeled data. We solve this problem in three steps, illustrated in Fig. 2. ❶ First we train a zero-shot (ZS) QoR predictor using prior synthesis data; as we show, it is sufficient to use *small* benchmarks to train this ZS model. ❷ Next, we fine-tune the ZS model using a small number of smartly selected synthesis runs of the AIG being optimized; and ❸ we deploy the fine-tuned QoR predictor within a black-box SA optimizer.

B. Zero-Shot QoR Predictor (❶)

We assume that a design house has access to data from previously synthesized designs or can generate data by running synthesis on *small* designs. We developed OpenABC-D [15]), an open-source automated synthesis data generation framework to generate data and train our ZS predictor.

In this work, we adopt a GCN-based architecture for the predictor (Fig. 1); GCNs have been used in RL-based approaches as well [7], [8], but with different architectures. Our predictor architecture is shown in Fig. 1(a) and contains two parallel paths. The first path takes an AIG as input and creates an

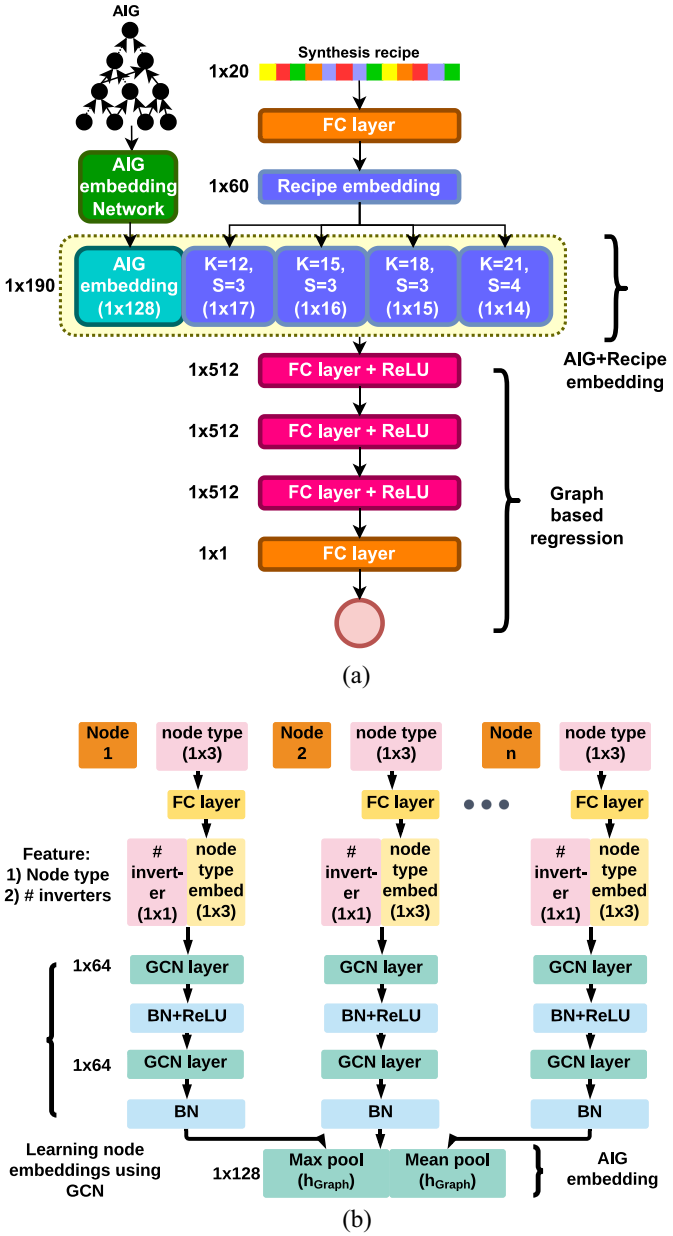


Fig. 1. Architecture for QoR prediction. GCN: Graph convolutional layer. *FC*: fully connected layers. *BN*: Batch normalization layer. (a) Network architecture. (b) AIG embedding network.

AIG embedding of the network; the second path extracts features from a synthesis recipe and outputs a "synthesis recipe embedding." We now describe the "AIG embedding" network and recipe embedding network:

1) AIG Embedding: We use GCN to represent the original AIG graph. GCN can learn and extract key features from DAG structures, particularly subgraph structures that can be optimized via various synthesis transformations. GCNs generalizes our model over different AIGs structures. For AIG embedding, we consider two features for each node: 1) type and 2) # inverter in fan-in. The type of node can be primary input (PI), primary output (PO), or an internal node. As shown in Fig. 1(a), we encode node type and pass it via a linear layer. The initial node level features pass through two consecutive

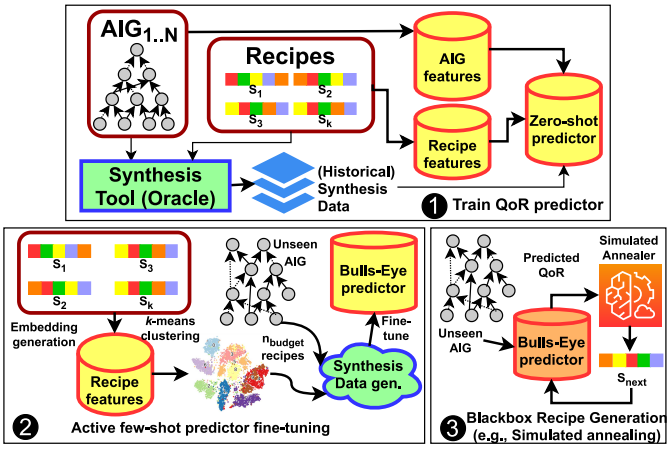


Fig. 2. Bulls-Eye framework.

GCN layers aggregating local neighborhood information based on connectivity. We use batch-norm followed by ReLU operation after each GCN layer. The graph-level embedding is a readout, like a global pool operation on all node embeddings. We choose concatenation of average pooling and max pooling to produce the AIG embedding h_{AIG}. The embeddings are defined as follows:

$$h_u^k = \sigma\left(W_k \times \sum_{u \cup N(u)} \frac{h_u^{k-1}}{\sqrt{N(u)} \times \sqrt{N(v)}} + b_k\right), k \in [1, 2] \tag{3}$$

$$h_{\text{AIG}} = \frac{1}{|V|} \sum_{u \in V} h_u^k \| \max_u h_u^k \tag{4}$$

where h_u^k is GCN embedding generated by kth layer of node u. W_k and b_k are GCN parameters and σ is nonlinear ReLU activation. $N(\cdot)$ denotes 1-hop neighbors.

2) Synthesis Recipe Embedding: In our work, we consider a fixed-length synthesis recipe **L**. We numerically encode the recipe using available synthesis transformation and pass it through a linear layer. Next, a set of 1-D convolutional neural networks (CNNs) with different kernel length extract features to produce "synthesis recipe embedding" (h_{recipe}). Mathematically, it can be shown as follows:

$$h_{\text{recipe}} = \text{concat}_i\left(b_i + \sum_{l=j}^{j+M} s_{kl} W_i^l\right), i \in [1, 4] \tag{5}$$

where M is kernel length and (W_i,b_i) are filter parameters.

The Bulls-Eye ZS predictor is created by concatenating AIG and synthesis recipe embeddings followed by four fully connected layers that output a QoR prediction (Fig. 2). Assuming a training dataset, $\mathcal{D}_{\text{train}}$, comprising AIGs ($g_i$), synthesis recipes ($s_i$), and labels represented by number of nodes (y_i)—normalized to respective samples of design in the dataset—the graph-based regression learns a parameterized function $\hat{F}(\cdot; \theta_{ZS})$ by minimizing the loss function

$$\mathcal{L}(y, \hat{y}) = \frac{1}{|\mathcal{D}_{\text{train}}|} \sum_{(g_i, s_i) \in \mathcal{D}_{\text{train}}} \left\| y_i - \hat{F}(g_i, s_i; \theta_{ZS}) \right\|_2^2. \tag{6}$$

Algorithm 1: Active Learning-Based Model Fine-Tuning

Data: Unseen AIG (g_U); Zero-shot predictor: $\hat{F}(\cdot; \theta_{ZS})$; synthesis recipes ($S$): $\{s_i, i \in [1, 1500]\}$, Synthesis budget runs: n_{budget}

Result: Active learning aided QoR predictor: $\hat{F}(\cdot; \theta_{FT}^A)$

Create recipe embedding $h_S \leftarrow \{h_{s_i} \forall s_i \in S\}$ using $\hat{F}(\cdot, \theta_{ZS})$.

$S_{cluster} \leftarrow k$-means clustering using n_{budget} heads on h_S.

Synthesize and generate labeled data

$D_U \leftarrow \{(g_U, s_k, y_k) | s_k \in S_{cluster}, |S_{cluster}| = n_{budget}\}$.

Fine-tune learnable parameters

$\theta_{FT}^A \leftarrow \theta_{ZS} - \alpha \nabla_\theta \mathcal{L}(\theta, \mathcal{D}_U)$

C. Fine-Tuning QoR Predictor (❷)

One can use the ZS model to predict QoR of different synthesis recipes on a *new* AIG. However, the accuracy of the model might be low if the features of the *new* AIG are different from prior data. We propose to *fine-tune* the ZS model using a **limited** number, n_{budget}, of synthesis runs on the new AIG, where n_{budget} is specified by the designer depending on the amount of time they are willing to spend.

Fine-tuning the ZS predictor avoids the need to train a model for each new design, which was a critical roadblock for prior work. This strategy has been successful in a variety of other domains, for example, image classification [16], but has not been explored for ML-guided synthesis. The next question we seek to answer is: *How should the synthesis budget n_{budget} be used?*

We explore two solutions. The first solution, which we refer to as **FT+R** fine-tunes the ZS model using *randomly* selected synthesis recipes. However, the aim of the designer is to pick the best synthesis recipes giving wide variations in the synthesis output. This problem can be mapped similarly to an *active learning* problem where label generation is a costly operation and therefore needs careful picking of data points. Inspired by the active learning approach [17], we propose a smarter solution, **FT+A**. Post training we have $|S|$ synthesis recipe embeddings, each having a dimension as shown in 5. We cast picking the most informative synthesis recipes as a *clustering problem* and we *cluster* the embeddings of the synthesis recipes used to train the ZS model into n_{budget} clusters (recall that each recipe has its own embedding denoted by h_{recipe}). We pick the cluster heads as our candidates for synthesis (instead of randomly picking recipes in FT + R). The intuition behind this idea is that the recipe cluster heads represent maximally diverse and informative points in the solution space. We synthesize the unseen AIG using the recipe cluster heads, generate labels, and fine-tune $\hat{F}(\cdot; \theta_{ZS})$ to produce a high-quality QoR predictor.

D. Black-Box Optimization (❸)

Although Bulls-Eye (the ZS and fine-tuned predictors) can be used in a wide variety of black-box approaches (e.g., evolutionary algorithms), we focus on SA adopted in [18]. A good

Algorithm 2: SA Recipe Generator

Data: Initial temperature ($T_{initial}$); Max. oracle queries: $Q_{\max}$; Fine-tuned QoR predictor $\hat{F}(\cdot; \theta^{A}_{FT})$, Design: ($AIG_U$)
Result: Node optimized synthesis recipe: $s^* \in M^L$
$T \leftarrow T_{initial}$, $Q \leftarrow 0$, $s^* \leftarrow$ **RANDOMRECIPE()**
while $Q < Q_{\max}$ **do**
  $s_{next} \leftarrow$ **NEIGHBOUR**(T, s^*) ▷ Neighbour recipes
  $\Delta E \leftarrow$ **ENERGY**$(s_{next}, \hat{F}(\cdot; \theta^{A}_{FT})) -$ **ENERGY**$(s^*, \hat{F}(\cdot; \theta^{A}_{FT}))$ ▷ Energy $\downarrow \Rightarrow$ no. of nodes $\downarrow$
  if $\Delta E < 0$ *or* **RANDOM()** $<$ **ACCEPT**$(T, \Delta E)$ **then**
    $s^* \leftarrow s_{next}$
  $T \leftarrow$ **COOLING**(T, s^*); $Q \leftarrow Q + 1$ ▷ Annealing schedule
return s^* ▷ Best synthesis recipe

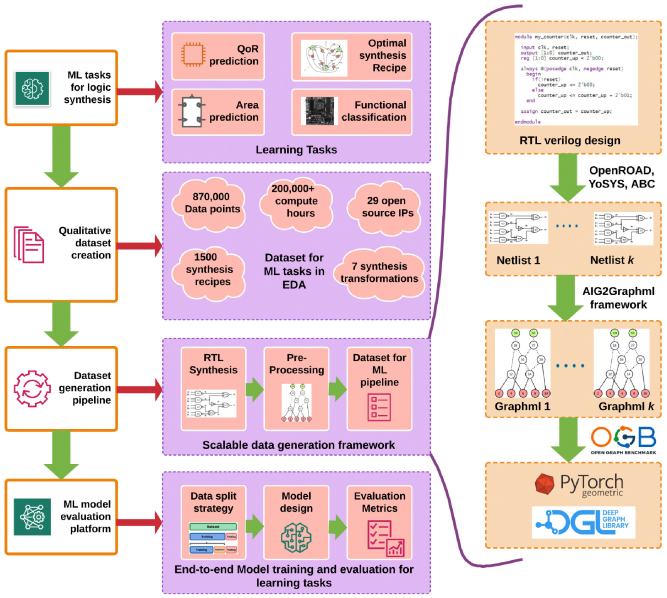


Fig. 3. OpenABC-D framework.

QoR predictor provides fast and accurate feedback to the optimizer by estimating the QoR of a synthesis recipe for an AIG. As we will show in our experimental evaluation, QoR predictor aided SA can substantially explore the synthesis recipe space in a given time span. Algorithm 2 outlines the flow for generating synthesis recipes. The terminology in Algorithm 2 is based on the SA literature; readers can peruse [18] for more background.

V. Data Generation and Analysis

Access and reproducibility are key to advancing ML research in EDA. We focus on open-source EDA platforms. We develop OpenABC-D (Fig. 3) as an end-to-end large-scale data generation framework. We apply OpenABC-D framework on open-source designs to generate the dataset used to train our Bulls-Eye predictor.

TABLE II
Characteristics of Open Source Designs (Unoptimized). PI—Primary Inputs and PO—Primary Outputs

Benchmark	Characteristics			
	PI	PO	Nodes	Depth
spi [19]	254	238	4219	35
i2c[19]	177	128	1169	15
ss_pcm[19]	104	90	462	10
usb_phy[19]	132	90	487	10
sasc[19]	135	125	613	9
wb_dma[19]	828	702	4587	29
simple_spi[19]	164	132	930	12
pci[19]	3429	3157	19547	29
ac97_ctrl[19]	2339	2137	11464	11
mem_ctrl[19]	1187	962	16307	36
des3_area[19]	303	64	4971	30
aes[19]	683	529	28925	27
sha256[20]	1943	1042	15816	76
fir[20]	410	351	4558	47
iir[20]	494	441	6978	73
tv80[19]	636	361	11328	54
fpu[21]	632	409	29623	819
dynamic_node[22]	2708	2575	18094	33
apex1[23]	45	45	1577	14
bc0[23]	26	11	1592	31
c1355[24]	41	32	512	29
c5315[24]	178	123	1613	38
c6288[24]	32	32	2337	120
c7552[24]	207	107	2198	30
dalu[23]	75	16	1735	35
i10[4]	257	224	273	58
k2[23]	45	45	2289	22
mainpla[23]	27	54	5346	38
div[25]	128	128	57247	4372
log2[25]	32	32	32060	444
max[25]	512	130	2865	287
multiplier[25]	128	128	27062	274
sin[25]	24	25	5416	225
sqrt[25]	128	64	24618	5058
square[25]	64	128	18484	250
aes_xcrypt[26]	1975	1805	45840	43
aes_secworks[27]	3087	2604	40778	42
bp_be[28]	11592	8413	82514	86
wb_conmax[19]	2122	2075	47840	24
ethernet[19]	10731	10422	67164	34
jpeg[19]	4962	4789	114771	40
tiny_rocket[22]	4561	4181	52315	80
picosoc[21]	11302	10797	82945	43
vga_lcd[19]	17322	17063	105334	23
sixteen[25]	117	50	16216836	140

A. Data Generation

We use 44 open-source designs with a wide range of functions. Since there is no dataset that represents many classes of designs (like the ImageNet dataset [16]), we hand-curated designs of different functions from MIT LL labs CEP [20], OpenCores [19], IWLS [29], ISCAS [24], [30], MCNC [23], and EPFL benchmarks [25]. These benchmarks include complex and functionally diverse compared to ISCAS [24], [30] and EPFL [25] benchmarks used in prior work. In the context of EDA, functionally diverse IPs should have diverse AIG structures (e.g., tree-like, balanced, and skewed) that mimic the distribution of real designs. Table II summarizes the structural characteristics of the data before synthesis (without optimizations). These designs are functionally diverse—bus communication protocols, computing processors, digital signal processing cores, crypto accelerators, and system controllers.

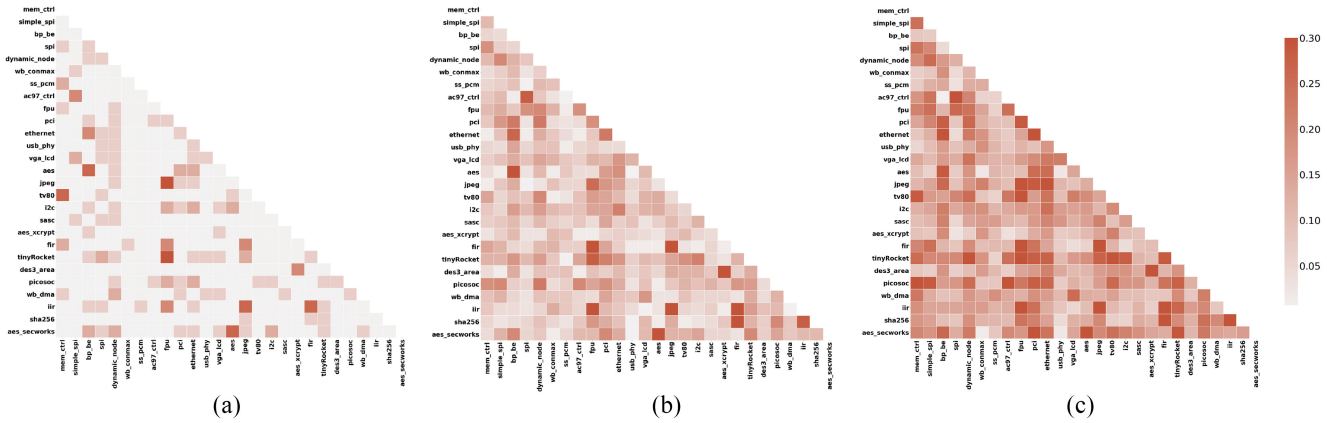


Fig. 4. Correlation plots amongst top $k\%$ synthesis recipes for IPs. Darker colors indicate more similarity between top recipes for IP pairs. (a) Top 1%. (b) Top 5%. (c) Top 10%.

We prepared $K = 1500$ synthesis recipes each $L = 20$ long, with a mix of `rewrite`, `resubstitute`, `refactor`, `rewrite -z`, `resub -z`, `refactor -z`, and `balance` transformations, consistent with prior works [6], [7], [8]. The synthesis recipes were prepared by randomly sampling from the set of synthesis transformations, assuming a uniform distribution. We ran synthesis with ABC using server-grade Intel processors for 200 000+ hours to generate the labeled data. We analyzed the top k synthesis recipes (in terms of node optimizations) by varying $k = 15$, 75, and 150 (1% to 10%). We found that the commonality of top recipes across any two designs is less than 30% (Fig. 4). Thus, AIG structures of designs are diverse and synthesis recipes perform differently across them.

B. Preprocessing for ML

This stage involves preparing the circuit data for use with any ML framework. We use PyTorch-geometric APIs to create data samples using AIG graphs, synthesis recipes, and the number of nodes and delay of synthesized AIGs. The dataset is available using a customized data loader for easy handling for preprocessing, labeling, and transformation. We create a script that helps partition the dataset (e.g., into train/test) based on user-specified learning tasks.

VI. Experimental Setup and Evaluation

Bulls-Eye illustrates fine-tuning, active learning, and scalability for ML-guided logic synthesis. We focus our experimental evaluation on characterizing Bulls-Eye in these dimensions.

A. Setup

We evaluate the efficacy of the Bulls-Eye predictor and the quality of the recipes found using Bulls-Eye on OpenABC-dataset. In our work, we have trained our networks for two target QoR labels: 1) normalized number of nodes post-synthesis and 2) normalized post-tech mapping area. We have used the NanGate 45-nm standard cell library for obtaining the area labels. We compare our method with prior SOTA ML approaches against the target QoR of normalized nodes post-synthesis. We emphasize that our work scales to larger designs compared to prior ML-based work.

For assessing the quality of a recipe, we use the following metrics for comparison. These metrics provide a qualitative indicator of the QoR metric optimized and relative improvement compared to baselines.

1) *Reduction Factor (%):* This factor depicts the improvement of QoR metric post-synthesis (e.g., nodes, area) to the QoR metric of the original netlist. This is denoted by

$$rf(\%) = \frac{N_{\text{original}} - N_{\text{synthesized}}}{N_{\text{original}}} * 100\%. \tag{7}$$

2) *Relative Improvement (%):* This metric measures the relative performance of the reduction factor of a proposed technique compared to the baseline technique. This is denoted by

$$\text{RI}(\%) = \frac{rf_{\text{proposed}} - rf_{\text{baseline}}}{rf_{\text{baseline}}} * 100\%. \tag{8}$$

We use these metrics to compare the quality of synthesis outputs using recipes generated by SA with Bulls-Eye QoR predictor. As an intermediate result, we evaluate the quality of the Bulls-Eye QoR predictors. We use a reference set of 1500 recipes and corresponding synthesis results as the ground-truth QoR. We perform standard normalization for each benchmark across 1500 ground truth labels. From this reference set, we draw the train/test recipe splits for evaluation.

In our experiment, we train the ZS predictor (Table III) on 18 designs ($\leq 30k$ nodes, upper half of Table II), and fine-tune and evaluate it on 28 designs (ten large designs with 41k−16M nodes) from the OpenABC dataset and EPFL benchmarks [25]. We evaluate the predictor on designs from prior works for comparisons to SOTA.

To train our models, we use mean-square error (MSE) as the loss function and normalized QoR metric as labels (normalized across each benchmark). The hyperparameters used include Adam optimizer, the learning rate of 0.01, and batch size of 64. During fine-tuning, we set the learning rate to 0.01 for fully connected layers and 0.001 for GCN and 1-D convolution layers. We perform experiments on a 2.9-GHz Intel Xeon

TABLE III
GCN ARCHITECTURE. I: INPUT DIMENSION, L1, L2: DIMENSION OF GCN LAYERS: F: # FILTERS, K: KERNELS, S: STRIDE, AND # L: #FC LAYERS

AIG Embedding				Recipe Encoding				FC Layers	
I	L1	L2	Pool	I	#f	k	s	# l	arch
4	64	64	Max+Mean	60	4	12,15,18,21	3	4	190-512-512-512-1

CPU with 384-GB RAM and 2 NVIDIA RTX8000 GPUs. We train the ZS model (❶) for 80 epochs with a runtime of about 12 h as a *one-time* effort. This relatively low one-time effort is because we only use small netlists to train the ZS predictor; surprisingly, we find it transfers well to new and unseen large netlists. Once the model ZS model is trained, it can be directly used or fine-tuned for *any* new design. We perform fine-tuning for 20 epochs that takes less than 800 s for each new design (up to 16M nodes) that we evaluate, including the large designs. We repeat our training experiment five times to generate five different models of each Bullseye predictor variant. This is performed to accommodate variation from various random sources.

B. Evaluating QoR Predictor

1) Experiment Design: We first examine how well the predictors enable accurate QoR prediction by comparing different variants of the Bulls-Eye predictor. We evaluate and compare their predictions on the reference set of randomly selected synthesis recipes (for which we have actual synthesis results). We capture this as the predictor's *Commonality factor*, which is defined as $([n(P \cap A)]/n(P))$, where A denotes top $k\%$ performing synthesis recipes (actual) and P denotes top $k\%$ synthesis recipes (predicted). As mentioned earlier, we use nine large benchmarks from the OpenABC-D framework, 19 benchmarks from SOTA works and EPFL benchmarks [25] for evaluation. The models are listed as follows.

1) *Standalone* QoR predictor is trained from scratch using n_{budget} samples from synthesizing the target design.
2) *ZS* QoR predictor is trained only on small designs from the OpenABC-D synthesis dataset.
3) *Fine-tuning + random* (FT + R) QoR predictor, which is fine-tuned on randomly chosen n_{budget} recipes and synthesis results as fine-tuning (calibration) data.
4) *Fine-tuning + active* (FT + A) QoR predictor, which is prepared by fine-tuning on synthesis data from n_{budget} most-informative recipes (generated from k-means clustering of the 1500 reference set recipe embeddings).

2) Analysis of Results: Table IV presents the performance of all QoR predictors using commonality factor as the metric across various top-$k\%$ values. We show the results of QoR predictors using post-synthesis nodes as the target QoR metric. For example, Table IV shows that Bulls-Eye using fine-tuning and active learning achieves an average score of 0.64 for top-25% on the *bp-be* design. This indicates 64% of top 25% performing synthesis recipes (out of 1450) on *bp-be* is common with top 25% synthesis recipes predicted by QoR predictor. Generally, we observe that fine-tuned QoR predictors yield better results (better commonality factor for every top $k\%$) compared to standalone and ZS predictors. A high commonality factor denotes that the model can infer better QoR achieving recipes successfully; pretrained predictors are indeed beneficial as the basis for fine-tuning QoR predictor.

TABLE IV
COMMONALITY AMONG TOP 25% RECIPES BETWEEN ACTUAL SYNTHESIS OUTPUT AND PREDICTIONS (↑ IS BETTER)

Design	Standalone	Bulls-Eye		
		Zero-shot	Fine-tune+Random	Fine-tune+Active
apex1	0.36 ± 0.01	0.36 ± 0.02	0.53 ± 0.02	0.56 ± 0.01
bc0	0.17 ± 0.02	0.44 ± 0.01	0.47 ± 0.02	0.48 ± 0.02
c1355	0.26 ± 0.01	0.32 ± 0.02	0.33 ± 0.01	0.35 ± 0.01
c5315	0.28 ± 0.02	0.28 ± 0.01	0.31 ± 0.02	0.36 ± 0.01
c6288	0.22 ± 0.01	0.26 ± 0.01	0.27 ± 0.02	0.29 ± 0.02
c7552	0.29 ± 0.02	0.35 ± 0.01	0.39 ± 0.01	0.42 ± 0.01
dalu	0.25 ± 0.02	0.48 ± 0.01	0.41 ± 0.01	0.41 ± 0.02
i10	0.37 ± 0.03	0.42 ± 0.02	0.54 ± 0.01	0.55 ± 0.01
k2	0.29 ± 0.03	0.20 ± 0.02	0.54 ± 0.01	0.55 ± 0.01
mainpla	0.21 ± 0.61	0.3 ± 0.04	0.51 ± 0.41	0.51 ± 0.41
div	0.27 ± 0.01	0.27 ± 0.02	0.29 ± 0.01	0.31 ± 0.01
log2	0.31 ± 0.01	0.27 ± 0.02	0.33 ± 0.02	0.35 ± 0.01
max	0.24 ± 0.03	0.27 ± 0.02	0.29 ± 0.01	0.31 ± 0.01
multiplier	0.36 ± 0.01	0.32 ± 0.02	0.43 ± 0.01	0.48 ± 0.01
sqrt	0.20 ± 0.02	0.30 ± 0.01	0.31 ± 0.01	0.33 ± 0.02
square	0.25 ± 0.61	0.30 ± 0.01	0.32 ± 0.01	0.33 ± 0.02
aes_secworks	0.31 ± 0.01	0.45 ± 0.02	0.57 ± 0.01	0.59 ± 0.02
aes_xcrypt	0.36 ± 0.01	0.36 ± 0.01	0.37 ± 0.01	0.40 ± 0.01
bp_be	0.35 ± 0.01	0.42 ± 0.01	0.61 ± 0.01	0.64 ± 0.02
ethernet	0.35 ± 0.01	0.38 ± 0.02	0.44 ± 0.01	0.46 ± 0.01
picosoc	0.40 ± 0.02	0.39 ± 0.01	0.41 ± 0.02	0.46 ± 0.01
jpeg	0.31 ± 0.02	0.45 ± 0.01	0.48 ± 0.01	0.50 ± 0.02
tinyRocket	0.33 ± 0.01	0.57 ± 0.02	0.60 ± 0.02	0.66 ± 0.01
vga_lcd	0.26 ± 0.01	0.36 ± 0.01	0.37 ± 0.02	0.38 ± 0.01
wb_conmax	0.29 ± 0.01	0.26 ± 0.01	0.25 ± 0.03	0.33 ± 0.01
chip_bridge	0.28 ± 0.02	0.34 ± 0.02	0.42 ± 0.02	0.45 ± 0.01
sixteen	0.13 ± 0.02	0.21 ± 0.02	0.24 ± 0.01	0.24 ± 0.02

C. QoR of Bulls-Eye Generated Recipes

1) Experiment Design: A good QoR predictor accurately predicts the quality of a synthesis recipe for a given design to provide good feedback in an optimization scheme. Bulls-Eye uses QoR predictors in lieu of actual synthesis runs as the backend engine ("evaluator") of an SA-based recipe generator. For a fair evaluation, we set two baselines to compare the quality of recipes generated using a QoR predictor: 1) k-repeat application of SA-generated recipe using actual synthesis runs as the evaluator and 2) resyn2*: repeated application of resyn2 synthesis recipe for same runtime budget as Bullseye. We set resyn2 as starting seed for SA and $k = 25$. We set a threshold of 1000 calls to the evaluator of SA and compare runtime speed up of QoR predictor versus actual synthesis. We compare Bulls-Eye on two aspects: 1) runtime speedup of SA using Bulls-Eye predictor instead of actual synthesis and 2) performance improvement of Bulls-Eye compared to resyn2* and other ML approaches [7], [8]. We set the initial temperature $T_{\text{start}} = 7250$ and distorted *Caunchy-Lorentz* distribution as visiting distribution in SA. We repeat SA for each setting ten times to report variations coming from random factors.

2) Analyzing Results Using Post-Synthesis Nodes as QoR: We compare post-synthesis number of nodes of synthesis output using Bulls-Eye generated recipes to our baselines and SOTA prior work [7], [8], [11] in Table V. This shows

TABLE V
REDUCTION FACTOR (%) OBTAINED USING POST-SYNTHESIS NUMBER OF NODES AS QOR

Design	Nodes	Reduction factor (%)													
		Baseline		SOTA			Conventional SA (SA_{1k})		Bulls-Eye(ZS)		Bulls-Eye(FT+R)			Bulls-Eye(FT+A)	
		resyn2 (x2)	resyn2*	[7]	[8]	[11]	$\mu \pm \sigma$	range	$\mu \pm \sigma$	range	$\mu \pm \sigma$	range		$\mu \pm \sigma$	range
apex1	1577	26.44	29.93	28.66	28.97	38.36	38.91 ± 0.61	35.89-40.67	32.06 ± 0.04	31.90-32.24	37.98 ± 0.41	36.71-39.27		$\mathbf{38.70 \pm 0.50}$	37.09-40.45
bc0	1592	44.40	45.03	48.49	48.05	**56.65**	53.70 ± 0.27	52.15-54.00	46.89 ± 0.29	46.00-47.78	51.13 ± 0.71	48.86-53.25		53.20 ± 0.22	52.54-53.86
c1355	512	23.04	23.24	23.42	23.43	23.40	23.46 ± 0.04	23.30-23.58	23.41 ± 0.02	23.32-23.47	23.43 ± 0.02	23.36-23.50		$\mathbf{23.44 \pm 0.02}$	23.36-23.54
c5315	1613	20.08	20.58	20.45	26.53	**22.81**	22.85 ± 0.02	22.75-22.94	22.50 ± 0.00	22.50-22.50	22.60 ± 0.01	22.55-22.64		22.78 ± 0.02	22.70-22.85
c6288	2337	19.98	19.98	19.98	19.98	**19.99**	19.99 ± 0.00	19.99-19.99	19.97 ± 0.00	19.97-19.97	$\mathbf{19.99 \pm 0.00}$	19.99-19.99		$\mathbf{19.99 \pm 0.00}$	19.99-19.99
c7552	2198	35.53	36.08	36.26	-	36.67	38.68 ± 0.07	38.48-38.89	37.42 ± 0.04	37.29-37.56	38.40 ± 0.02	38.22-38.57		$\mathbf{38.62 \pm 0.07}$	38.39-38.84
dalu	1735	39.37	39.6	42.31	40.58	46.10	47.05 ± 0.24	46.33-47.80	$\mathbf{46.15 \pm 0.27}$	45.28-47.95	46.06 ± 0.13	45.05-46.44		46.12 ± 0.43	44.82-47.72
i10	2273	25.43	25.43	27.63	-	27.27	27.62 ± 0.05	27.46-27.78	26.10 ± 0.08	25.85-26.34	26.94 ± 0.06	26.74-27.15		$\mathbf{27.59 \pm 0.05}$	27.42-27.76
k2	2289	44.82	46.48	47.53	47.66	50.54	51.03 ± 0.35	49.97-52.10	47.68 ± 0.07	47.46-47.90	49.91 ± 0.29	48.94-50.87		$\mathbf{50.99 \pm 0.34}$	49.81-52.02
mainpla	5346	34.12	34.46	35.69	35.3	**41.30**	40.98 ± 0.24	40.23-41.73	38.12 ± 0.33	37.13-39.09	40.06 ± 0.07	39.78-40.29		41.23 ± 0.08	40.95-41.50
div	57247	28.88	28.92	-	-	63.90	64.10 ± 0.03	64.00-64.20	60.71 ± 0.10	60.39-61.00	62.08 ± 0.02	62.01-62.14		$\mathbf{64.07 \pm 0.02}$	64.01-64.14
log2	32060	8.74	8.80	-	-	**9.60**	9.40 ± 0.04	9.27-9.52	8.84 ± 0.00	8.81-8.87	9.26 ± 0.01	9.23-9.29		9.39 ± 0.01	9.35-9.43
max	2865	1.19	1.19	-	-	1.27	1.45 ± 0.02	1.39-1.51	1.19 ± 0.00	1.19-1.19	1.28 ± 0.01	1.26-1.30		$\mathbf{1.30 \pm 0.01}$	1.28-1.32
multiplier	27062	9.27	9.27	-	-	10.06	10.10 ± 0.02	10.03-10.17	9.95 ± 0.01	9.91-10.00	10.0 ± 0.04	9.87-10.14		$\mathbf{10.08 \pm 0.02}$	10.02-10.14
sqrt	24618	21.05	21.05	-	-	21.96	22.31 ± 0.05	22.15-22.47	21.57 ± 0.14	21.12-22.01	21.98 ± 0.07	21.76-22.21		$\mathbf{22.20 \pm 0.04}$	22.07-22.32
square	18484	10.37	10.37	-	-	**14.46**	14.50 ± 0.02	14.42-14.57	12.85 ± 0.03	12.74-12.96	14.27 ± 0.03	14.18-14.36		14.42 ± 0.03	14.31-14.53
aes_secworks	40778	23.5	27.23	-	-	25.65	27.75 ± 0.06	27.55-27.94	27.61 ± 0.04	27.48-27.75	27.69 ± 0.03	27.59-27.79		$\mathbf{27.70 \pm 0.06}$	27.52-27.88
aes_xcrypt	45850	9.01	16.03	-	-	12.48	16.22 ± 0.04	16.09-16.35	16.16 ± 0.02	16.10-16.22	16.19 ± 0.01	16.15-16.23		$\mathbf{16.21 \pm 0.01}$	16.16-16.26
bp_be	82514	4.92	5.05	-	-	5.38	5.47 ± 0.11	5.25-5.78	5.14 ± 0.08	4.90-5.28	5.41 ± 0.04	5.28-5.54		$\mathbf{5.45 \pm 0.08}$	5.09-5.70
ethernet	67164	3.30	3.44	-	-	3.45	3.47 ± 0.01	3.45-3.49	3.45 ± 0.01	3.45-3.48	3.46 ± 0.01	3.45-3.48		$\mathbf{3.46 \pm 0.01}$	3.45-3.48
picosoc	82945	7.40	7.54	-	-	7.60	7.86 ± 0.06	7.66-8.07	7.32 ± 0.08	7.06-7.58	7.51 ± 0.10	7.19-7.83		$\mathbf{7.79 \pm 0.08}$	7.54-8.04
jpeg	114771	9.18	9.37	-	-	**11.30**	11.21 ± 0.02	11.12-11.30	11.10 ± 0.02	11.03-11.17	11.14 ± 0.01	11.10-11.19		11.19 ± 0.02	11.12-11.25
tinyRocket	52315	21.30	21.80	-	-	22.10	22.28 ± 0.05	22.12-22.45	22.00 ± 0.03	21.90-22.10	22.16 ± 0.03	22.00-22.27		$\mathbf{22.20 \pm 0.05}$	22.04-22.36
vga_lcd	105334	1.62	1.64	-	-	1.63	1.72 ± 0.01	1.69-1.74	1.67 ± 0.01	1.64-1.70	1.69 ± 0.01	1.68-1.71		$\mathbf{1.70 \pm 0.01}$	1.68-1.73
wb_conmax	47840	6.72	6.89	-	-	**10.22**	10.23 ± 0.07	10.01-10.45	9.91 ± 0.11	9.54-10.27	10.06 ± 0.08	9.80-10.32		10.15 ± 0.09	9.87-10.43
chip_bridge	124565	43.56	44.67	-	-	45.02	46.01 ± 0.05	45.83-46.17	44.54 ± 0.11	44.22-44.89	45.11 ± 0.08	44.87-45.41		$\mathbf{45.23 \pm 0.09}$	44.96-45.53
sixteen$^{\$}$	16216836	26.48	26.97	-	-	-	27.58 ± 0.15	27.10-28.01	26.97 ± 0.03	26.87-27.09	27.02 ± 0.03	26.91-27.12		$\mathbf{27.02 \pm 0.04}$	26.87-27.18
Time (in s)	-	5.338	3932.8	14946.4†	1387.9†	193.3‡	3932.8		529.6		1250.53			1333.91	

† Runtime scaled on our machine based on data provided by authors [8] for synthesis runs during training. Total training runtime will be higher than reported here.
$ Queries to synthesis oracle limited to 100 (instead of 1000) due to large run-time overhead (7 days). [11] threw runtime error on this benchmark.
‡ The framework is end-to-end integrated inside ABC logic synthesis framework.

synthesis output using Bulls-Eye generated recipes achieve very similar QoR compared to conventional SA with **3.7**× median runtime speedup and outperforms resyn2*. The reduction factor in **bold** denotes the top-performing recipe. Our results show Bulls-Eye generated recipes using finetuning perform better than SOTA on 19 out of 27 benchmarks (Table V). Compared to [7] and [8], our best predictor (FT + A) achieves better *rf %* (**3%–10%**) with a **11.20**× and **1.04**× runtime speed-up, respectively. Our (FT + R) predictor achieves better *rf %* than [7] and [8] (**2.7%** and **1.35%**) with **12**× and **1.10**× runtime speed-up, respectively. Compared to [7] and [8], the ZS predictor *ZS* obtains almost similar QoR, however achieving a huge runtime speed-up over prior work (**45**× and **4**×, respectively). This shows that our ZS predictor is helpful for prediction on unseen netlists. The prediction quality improved through fine-tuning yields better recipes, i.e., recipes found using the active learning guided fine-tuned (FT + A) model perform better than those found by the ZS and random recipe augmented fine-tuned (FT + R) QoR predictors. The result shows the effectiveness of Bulls-Eye on **large** benchmarks ($\geq 40k$ nodes) where [7], [8] fail. We also compare QoR of our work with [11] which presents multiarm bandits-based logic optimization. We found Bullseye (FT + A) achieves better QoR on **6 out of top 10** designs listed in Table V) albeit with **6.92**× extra runtime overhead. The reason can be attributed to the low-level tight integration of [11] with ABC framework making it extremely runtime efficient.

Table VI shows runtime complexity and speedup of Bulls-Eye guided recipe generation. We present relative improvement and timing speedup of Bulls-Eye in Fig. 5. Benchmark *div* achieved > 100% RI compared to resyn2* indicating that Bulls-Eye generated recipe optimized twice the number of

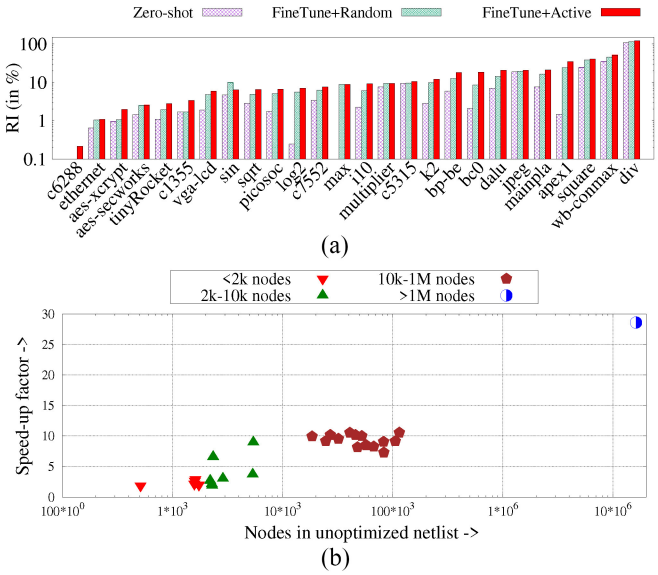


Fig. 5. Bullseye performance. (a) RI with respect to resyn2*. (b) Speed-up compared to conventional SA for post-synthesis nodes as QoR.

nodes compared to resyn2 applied for the same time duration. Further, runtime speedup improves with more nodes in initial AIG. All designs having $\geq 10k$ nodes in initial AIG achieved more than **8**× speed-up with similar QoR quality compared to SA_{1k}. To illustrate the faster recipe space search that comes from using Bulls-Eye QoR predictors, we chart the QoR progression over time in Fig. 6. The graphs show Bulls-Eye can generate superior quality synthesis recipes quickly compared to the conventional SA approach that uses actual synthesis runs. The graph for FT + A includes an initial time penalty

TABLE VI
TIMING COMPLEXITY OF BASELINE AND BULLS-EYE FRAMEWORK. FT: TIME FOR FINETUNING THE MODEL. *SA*: TIME FOR SA. *synth*: TIME FOR $k = 25$ REPEATED APPLICATION OF SYNTHESIS RECIPE

Design	Time taken (s)															Speedup		
	resyn2 (x2)	Baseline (SA_{1k})			ZS			FT+R				FT+A				ZS	FT+R	FT+A
		SA	synth	total	SA	synth	total	FT	SA	synth	total	FT	SA	synth	total			
apex1	0.4	216.5	13.2	229.7	35.5	11.3	46.8	48.3	30.2	13.5	92.0	53.1	37.6	15.9	106.6	5.0	2.5	2.2
bc0	0.3	235.2	11.2	246.4	36.9	9.8	46.7	42.2	30.2	12.3	84.7	46.5	37.5	9.1	93.0	5.3	2.9	2.7
c1355	0.2	135.3	5.4	140.7	37.5	5.5	43.0	40.1	28.0	4.9	73.0	44.1	26.1	5.2	75.4	3.3	1.9	1.9
c5315	0.5	287.1	16.5	303.6	32.6	17.5	50.1	49.6	33.0	19.1	101.7	54.5	32.3	16.5	103.3	6.1	3.0	2.9
c6288	1.5	928.3	60.5	988.8	23.7	57.4	81.1	51.2	38.7	61.6	151.5	56.4	23.5	70.5	150.3	12.2	6.5	6.6
c7552	0.5	358.8	17.0	375.8	29.3	18.3	47.6	90.6	39.2	21.1	150.9	99.6	23.2	17.7	140.5	7.9	2.5	2.7
dalu	0.4	267.7	9.2	276.9	27.7	9.2	36.9	74.5	40.0	13.9	128.4	82.0	43.1	12.7	137.7	7.5	2.2	2.0
i10	0.5	342.7	21.4	364.0	33.7	17.0	50.7	98.6	35.2	20.5	154.3	108.4	40.4	23.2	172.0	7.2	2.4	2.1
k2	0.3	308.7	13.6	322.3	38.1	12.1	50.2	84.5	35.7	14.8	134.9	92.9	62.2	15.1	170.2	6.4	2.4	1.9
mainpla	0.8	636.7	47.9	684.6	30.2	46.4	76.6	99.4	30.7	49.2	179.2	109.3	29.8	45.8	184.9	8.9	3.8	3.7
div	12.7	6554.4	291.9	6846.2	115.6	250.2	365.8	104.7	171.7	280.2	556.6	115.2	379.4	309.5	804.1	18.7	12.3	8.5
log2	11.0	7614.2	587.5	8201.7	67.5	409.4	476.9	110.6	70.3	569.4	750.3	121.6	165.0	571.0	857.6	17.2	10.9	9.6
max	0.7	397.9	22.5	420.4	30.2	17.0	47.2	78.7	30.2	23.4	132.3	86.5	27.3	25.3	139.2	8.9	3.2	3.0
multiplier	8.2	5359.1	487.2	5846.3	58.7	394.8	453.5	97.9	117.9	457.4	673.2	107.7	58.4	408.2	574.3	12.9	8.7	10.2
sin	1.9	1216.4	342.6	1559.0	63.4	85.6	149.0	52.4	34.3	110.8	197.5	57.6	26.5	89.8	173.9	10.5	7.9	9.0
sqrt	7.4	4460.3	340.6	4800.9	39.9	294.1	334.0	110.5	61.7	340.0	512.2	121.5	55.1	346.6	523.2	14.4	9.4	9.2
square	6.7	3913.0	345.6	4258.6	58.7	313.8	372.5	100.4	23.5	360.9	484.7	110.4	44.6	273.3	428.3	11.4	8.8	10.0
aes_secworks	6.2	5526.7	310.5	5837.2	77.2	282.4	359.6	101.2	138.8	333.8	573.9	111.4	94.4	348.2	553.9	16.2	10.2	10.5
aes_xcrypt	9.0	8293.0	489.8	8782.8	201.1	462.2	663.3	99.9	128.9	504.7	733.5	109.9	296.3	457.9	864.0	13.2	12.0	10.2
bp_be	18.9	14890.4	1457.9	16348.2	219.5	1212.3	1431.8	145.6	503.5	1523.9	2172.9	160.1	396.3	1247.8	1804.2	11.4	7.5	9.1
ethernet	8.26	7014.6	514.2	7528.9	301.1	454.7	755.8	157.0	278.0	525.6	960.6	172.7	226.9	509.8	909.4	10.0	7.8	8.3
jpeg	39.7	27934.4	2220.2	30154.7	257.4	2154.1	2411.4	175.5	480.8	2231.6	2887.9	193.1	515.6	2147.9	2856.6	12.5	10.4	10.6
picosoc	10.6	8714.8	614.0	9328.8	273.6	583.2	856.9	112.4	462.4	623.6	1198.3	123.6	587.4	567.5	1278.5	10.9	7.8	7.3
tinyRocket	8.1	7366.6	536.6	7903.3	195.4	517.2	712.6	101.2	310.4	554.1	965.7	111.4	154.5	524.7	790.6	11.1	8.2	10.0
vga_lcd	15.6	13397.8	1101.2	14498.9	256.3	923.8	1180.0	180.6	410.1	1079.1	1669.8	198.6	412.2	973.2	1584.0	12.3	8.7	9.2
wb_conmax	6.6	6496.6	469.9	6966.5	103.8	465.2	569.0	82.0	301.2	417.2	800.5	90.2	327.6	436.1	853.9	12.2	8.7	8.2
chip_bridge	45.2	36917.6	2740.2	39657.8	285.4	2287.1	2572.5	188.4	508.8	2212.6	2909.8	187.1	537.6	2287.9	3012.6	15.4	13.6	13.1
sixteen	5752.3	$4.75x10^6$	$1.43x10^5$	$4.89x10^6$	1240.5	$1.55x10^5$	$1.55x10^5$	768.2	1387.6	$1.39x10^5$	$1.60x10^5$	798.4	1562.3	$1.48x10^5$	$1.71x10^5$	31.5	30.6	28.6

TABLE VII
REDUCTION FACTOR (%) OBTAINED USING POST-TECHMAPPING AREA AS QOR

Design	Reduction factor (%)									
	Baseline		Conventional SA (SA_{1k})		Bulls-Eye(ZS)		Bulls-Eye(FT+R)		Bulls-Eye(FT+A)	
	resyn2*	[11]	$\mu \pm \sigma$	range	$\mu \pm \sigma$	range	$\mu \pm \sigma$	range	$\mu \pm \sigma$	range
aes_secworks	14.68	16.77	24.00 ± 0.03	23.91-24.09	22.61 ± 0.04	22.25-22.74	22.69 ± 0.02	22.54-22.81	$\mathbf{22.69 \pm 0.06}$	22.52-22.93
aes_xcrypt	7.35	**12.10**	12.05 ± 0.03	11.96-12.14	10.98 ± 0.10	10.65-11.31	11.76 ± 0.05	11.55-11.92	11.81 ± 0.05	11.69-11.99
bp_be	9.02	18.31	20.21 ± 0.01	20.16-20.25	16.78 ± 0.06	16.57-17.00	18.42 ± 0.04	18.29-18.60	$\mathbf{19.10 \pm 0.03}$	18.90-19.25
ethernet	3.23	**7.79**	7.67 ± 0.25	6.89-8.20	4.32 ± 0.28	3.43-5.33	6.89 ± 0.06	6.69-7.20	7.50 ± 0.10	6.93-7.83
jpeg	6.97	7.16	14.21 ± 0.17	13.71-14.77	7.10 ± 0.01	7.05-7.15	10.14 ± 0.01	10.05-10.18	$\mathbf{12.19 \pm 0.05}$	12.00-12.40
picosoc	9.25	**15.49**	15.42 ± 0.03	15.27-15.60	10.16 ± 0.02	10.00-10.23	13.45 ± 0.07	13.12-13.77	15.32 ± 0.02	15.05-15.44
tinyRocket	18.82	21.47	22.00 ± 0.01	21.95-22.05	20.05 ± 0.04	19.85-20.25	21.34 ± 0.03	21.10-21.47	$\mathbf{21.80 \pm 0.01}$	21.70-21.88
vga_lcd	2.15	3.15	4.01 ± 0.02	3.90-4.10	2.67 ± 0.03	2.50-2.77	3.01 ± 0.02	2.87-3.10	$\mathbf{3.68 \pm 0.03}$	3.55-3.80
wb_conmax	3.64	**6.75**	6.12 ± 0.02	6.00-6.25	4.67 ± 0.07	4.42-4.90	5.21 ± 0.06	5.01-5.39	5.98 ± 0.05	5.82-6.14
div	34.00	62.14	69.40 ± 0.03	69.29-69.58	64.50 ± 0.05	64.34-64.66	67.88 ± 0.07	67.68-68.09	$\mathbf{68.91 \pm 0.03}$	68.80-69.00
multiplier	7.11	4.65	8.39 ± 0.02	8.33-8.46	7.59 ± 0.03	7.48-7.69	7.99 ± 0.04	7.86-8.14	$\mathbf{8.10 \pm 0.02}$	8.04-8.17
sqrt	18.56	22.61	45.93 ± 1.01	44.88-46.97	39.87 ± 0.03	39.76-40.00	41.98 ± 0.12	41.40-42.35	$\mathbf{43.54 \pm 0.11}$	43.21-43.87
square	-12.32	4.25	6.05 ± 0.02	5.98-6.11	4.87 ± 0.05	4.69-5.03	5.32 ± 0.07	5.92-6.10	$\mathbf{6.00 \pm 0.03}$	5.92-6.10
log	-0.02	1.46	10.07 ± 0.05	9.91-10.22	5.98 ± 0.14	5.45-6.39	6.19 ± 0.22	5.82-6.67	$\mathbf{9.67 \pm 0.05}$	9.50-9.83

from synthesis data generation and fine-tuning the model; after this, the models quickly explore the recipe space due to fast inference by the proxy models.

3) Analyzing Results Using Post-Tech Mapped Area as QoR: We now compare area post technology mapping and synthesis output using Bulls-Eye generated recipes to our baselines and SOTA prior work [11] in Table VII. Here, we use the Bulls-Eye predictor trained on the area QoR metric. FT + *A* generated recipe achieves around 0.93 times the reduction factor obtained by the recipe generated using actual synthesis. This shows that the Bulls-Eye predictor can significantly cut down runtime overhead by generating qualitative synthesis recipes using a black-box optimizer (e.g., SA). In Table VII, we show Bullseye predictors outperform QoR obtained compared resyn2*. We also compare our results with prior work [11]. The data shows FT+*A* achieves better QoR on **10 out of 14** designs compared to [11] with considerably better QoR (**up to 20%**) on *aes_secworks*, *jpeg*, *div*, *multiplier*, *sqrt*, and *log*.

VII. CONCLUSION

Bulls-Eye is a solution for ML-guided synthesis recipe generation that scales to large designs. Bulls-Eye uses few-shot learning and active learning to build upon models pretrained on historical synthesis data, given a limited synthesis run budget for new, unseen designs. Our framework generates better synthesis recipes compared to SOTA, with 3.7× median

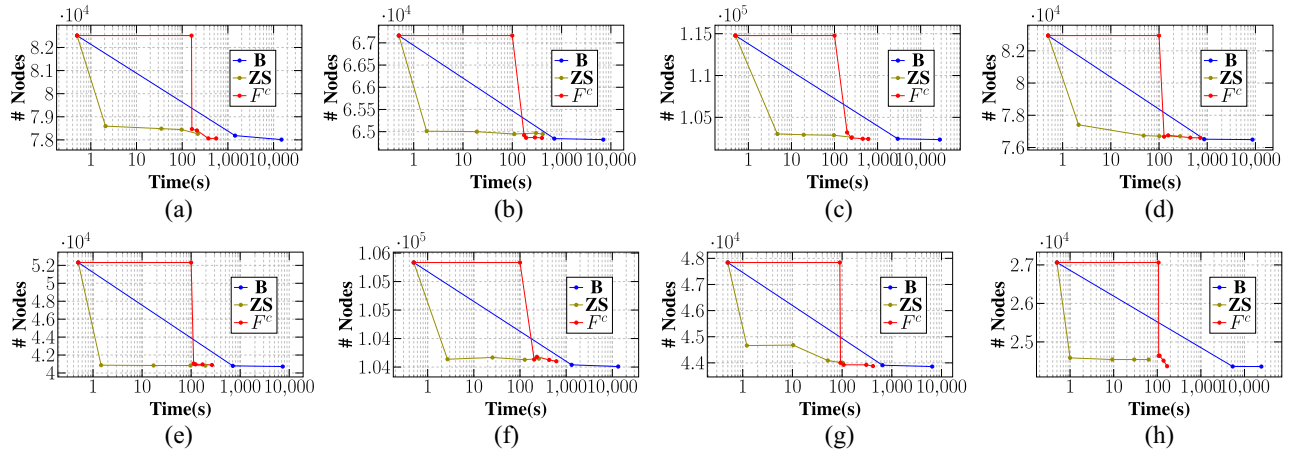


Fig. 6. QoR at various times from recipes by SA using different evaluators: actual synthesis (B) and proxy predictors (ZS and FT+A). (a) bp_be. (b) Ethernet. (c) jpeg. (d) picosoc. (e) tinyRocket. (f) vga_lcd. (g) wb_conmax. (h) Multiplier.

runtime speed-up and up to 30× speed-up (on benchmarks ≥ 1M nodes) compared to conventional black-box approaches. Our code and dataset are available at: https://github.com/NYU-MLDA/OpenABC.

REFERENCES

[1] A. Mirhoseini et al., "A graph placement methodology for fast chip design," *Nature*, vol. 594, no. 7862, pp. 207–212, 2021.

[2] W. L. Neto, M. Austin, S. Temple, L. Amaru, X. Tang, and P.-E. Gaillardon, "LSOracle: A logic synthesis framework driven by artificial intelligence: Invited paper," in *Proc. ACM/IEEE Int. Conf. Comput.-Aided Des.*, New York, NY, USA, 2019, pp. 1–6.

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

[4] R. Brayton and A. Mishchenko, "ABC: An academic industrial-strength verification tool," in *Proc. Int. Conf. Comput.-Aided Verif.*, 2010, pp. 24–40.

[5] C. Yu, H. Xiao, and G. D. Micheli, "Developing synthesis flows without human knowledge," in *Proc. ACM/IEEE Design Autom. Conf.*, New York, NY, USA, 2018, pp. 1–6.

[6] A. Hosny, S. Hashemi, M. Shalan, and S. Reda, "DRiLLS: Deep reinforcement learning for logic synthesis," in *Proc. IEEE Asia South Pac. Des. Autom. Conf.*, New York, NY, USA, 2020, pp. 581–586.

[7] K. Zhu, M. Liu, H. Chen, Z. Zhao, and D. Z. Pan, "Exploring logic optimizations with reinforcement learning and graph convolutional network," in *Proc. ACM/IEEE Workshop Mach. Learn. CAD*, Reykjavik, Iceland, 2020, pp. 145–150.

[8] Y. V. Peruvemba, S. Rai, K. Ahuja, and A. Kumar, "RL-guided runtime-constrained heuristic exploration for logic synthesis," in *Proc. ACM/IEEE Int. Conf. Comput.-Aided Des.*, Munich, Germany, 2021, pp. 1–9.

[9] M. Welling and T. N. Kipf, "Semi-supervised classification with graph convolutional networks," in *Proc. Int. Conf. Learn. Represent. (ICLR)*, 2017, pp. 1–14.

[10] W. Haaswijk et al., "Deep learning for logic optimization algorithms," in *Proc. IEEE Int. Symp. Circuits Syst.*, Florence, Italy, 2018, pp. 1–4.

[11] C. Yu, "FlowTune: Practical multi-armed bandits in Boolean optimization," in *Proc. ACM/IEEE Int. Conf. Comput.-Aided Des.*, San Diego, CA, USA, 2020, pp. 1–9.

[12] D. Silver et al., "Mastering the game of Go without human knowledge," *Nature*, vol. 550, no. 7676, pp. 354–359, 2017.

[13] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet classification with deep convolutional neural networks," in *Proc. Int. Conf. Adv. Neural Inf. Process. Syst.*, vol. 25, 2012, pp. 1097–1105.

[14] Y. Wang, Q. Yao, J. T. Kwok, and L. M. Ni, "Generalizing from a few examples: A survey on few-shot learning," *ACM Comput. Surv.*, vol. 53, no. 3, pp. 1–34, 2020.

[15] A. B. Chowdhury, B. Tan, R. Karri, and S. Garg. "OpenABC-D: A large-scale dataset for machine learning guided integrated circuit synthesis." 2021. [Online]. Available: https://github.com/NYU-MLDA/OpenABC

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

[17] Z. Bodó, Z. Minier, and L. Csató, "Active learning with clustering," in *Proc. Act. Learn. Exp. Des. Workshop*, 2011, pp. 127–139.

[18] Y. Xiang, D. Y. Sun, W. Fan, and X. G. Gong, "Generalized simulated annealing algorithm and its application to the Thomson model," *Phys. Lett. A*, vol. 233, no. 3, pp. 216–220, 1997.

[19] "Opencores hardware RTL designs." Accessed: May 20, 2021. [Online]. Available: https://opencores.org

[20] "MIT common evaluation platform(CEP)." 2020. [Online]. Available: https://github.com/mit-ll/CEP

[21] J. Balkind et al., "OpenPiton: An open source manycore research framework," *ACM SIGPLAN Notices*, vol. 51, no. 4, pp. 217–232, 2016.

[22] T. Ajayi et al., "OpenROAD: Toward a self-driving, open-source digital layout implementation tool chain," in *Proc. GOMACTECH*, 2019, pp. 1105–1110.

[23] S. Yang, "Logic synthesis and optimization benchmarks user guide: Version 3.0," MCNC, Research Triangle Park, NC, USA, Rep. 1991-IWLS-UG-Saeyang, 1991.

[24] F. Brglez and H. Fujiwara, "A neutral netlist of 10 combinational benchmark circuits and a target translator," in *Proc. IEEE Int. Symp. Circuits Syst.*, New York, NY, USA, 1985, pp. 1–10.

[25] L. Amarú, P.-E. Gaillardon, and G. De Micheli, "The EPFL combinational benchmark suite," in *Proc. IEEE Int. Workshop Logic Synthesis*, New York, NY, USA, 2015, pp. 1–5.

[26] X. J. Jiang. "AES 128/256-bit symmetric block cipher." 2019. [Online]. Available: https://github.com/crypt-xie/XCryptCore/tree/master/ciphers/aes

[27] J. Strömbergson and O. Kindgren. "AES 128/256-bit symmetric block cipher." 2022. [Online]. Available: https://github.com/secworks/aes

[28] "Black Parrot SoC." 2022. [Online]. Available: https://github.com/black-parrot/black-parrot

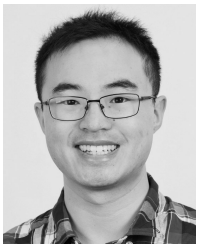
[29] C. Albrecht, "IWLS 2005 benchmarks," in *Proc. IEEE Int. Workshop Logic Synth.*, New York, NY, USA, 2005, pp. 1–18.

[30] F. Brglez, D. Bryan, and K. Kozminski, "Combinational profiles of sequential benchmark circuits," in *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)*, 1989, pp. 1929–1934.

Animesh Basak Chowdhury received the M.Tech. degree in computer science from Indian Statistical Institute, Kolkata, India, in 2016. He is currently pursuing the Doctoral degree with NYU Centre for Cybersecurity, Brooklyn, NY, USA.

Prior to joining the Ph.D. program, he spent three years as a Researcher with Tata Research Development and Design Centre, Pune, India. His research interests include machine learning for electronics design automation.

Mr. Chowdhury has won awards and recognition in International Software Verification and Testing Competitions.



Benjamin Tan (Member, IEEE) received the B.E. degree (Hons.) in computer systems engineering and the Ph.D. degree from the University of Auckland, Auckland, New Zealand, in 2014 and 2019, respectively.

He was a Professional Teaching Fellow with the Department of Electrical and Computer Engineering, University of Auckland in 2018. From 2019 to 2021, he was with New York University, Brooklyn, NY, USA, where he was a Postdoctoral Associate and then a Research Assistant Professor, affiliated with NYU Center for Cybersecurity. He is currently an Assistant Professor with the University of Calgary, Calgary, AB, Canada. His research interests include computer engineering, hardware security, and electronic design automation.

Dr. Tan is a member of ACM.

Ryan Carey received the B.S. degree in biomedical engineering from Worcester Polytechnic Institute, Worcester, MA, USA, in 2005, and the M.S. and Ph.D. degrees in neuroscience from Boston University, Boston, MA, USA, in 2009 and 2012, respectively.

He joined Qualcomm Technologies, Inc., San Diego, CA, USA, in 2013, where he currently serves as a Senior Staff Engineer. He worked on spiking neural networks, low-power processing of mobile sensor data, and has been granted eight U.S. patents. His research interests include application of machine learning to problems in computer-aided VLSI design and wireless networks.

Tushit Jain received the B.Tech. degree in electrical engineering and the M.Tech. degree in electrical engineering in 2003 from the Indian Institute of Technology Madras, Chennai, India.

He currently serves as a Sr. Director with Qualcomm Technologies, Inc., San Diego, CA, USA. He has worked on several different areas in semiconductor industry, including wireless networking and VLSI design. His research interests include application of machine learning to problems in computer-aided VLSI design and wireless networks.

Ramesh Karri (Fellow, IEEE) received the B.E. degree in electrical and computer engineering from Andhra University, Visakhapatnam, India, in 1985, and the Ph.D. degree in computer science and engineering from the University of California at San Diego, San Diego, CA, USA, in 1993.

He is currently a Professor of Electrical and Computer Engineering with New York University (NYU), Brooklyn, NY, USA. He co-directs the NYU Center for Cyber Security. He has authored or coauthored more than 240 articles in leading journals and conference proceedings. His current research interests include hardware cybersecurity include trustworthy ICs; processors and cyber–physical systems; security-aware computer-aided design, test, verification, validation, and reliability; nano meets security; hardware security competitions, benchmarks, and metrics; biochip security; and additive manufacturing security.

Siddharth Garg received the B.Tech. degree in electrical engineering from IIT Madras, Chennai, India, and the Ph.D. degree in electrical and computer engineering from Carnegie Mellon University, Pittsburgh, PA, USA, in 2009.

He was an Assistant Professor with the University of Waterloo, Waterloo, ON, Canada, from 2010 to 2014. In 2014, he joined New York University, Brooklyn, NY, USA, as an Assistant Professor. His general research interest includes computer engineering, more particularly secure, reliable, and energy-efficient computing.

Dr. Garg was a recipient of the NSF CAREER Award in 2015.