Learning Based Presilicon Estimation of Design Area from Early EDA Metrics

Tarun Kholay[1][0009−0009−1183−2287], Anup Ashok Kedilaya[1][0009−0000−9213−4468], Aman Arora[2][0000−0003−2547−4424], Eugene B. John[3][0000−0001−9494−4894], Jaydeep P. Kulkarni[1][0000−0002−0258−6776], and Lizy K. John[1][0000−0002−8747−5214]

[1] The University of Texas at Austin, Austin, TX, USA
{tarunkholay, anup027, jaydeep, ljohn}@utexas.edu
[2] Arizona State University, Phoenix, AZ, USA
aman.kbm@asu.edu
[3] The University of Texas at San Antonio, San Antonio, TX, USA
eugene.john@utsa.edu

Abstract. In early stages of prototyping and acceleration, designers rely on vendor-specific and architecture-specific resource metrics (e.g., LUTs, ALMs, DSP blocks), which prevent direct cross-platform comparison and complicate design space exploration. We present MARU (Machine Learning-based Approaches for Resource-to-Area Estimation), a framework that predicts ASIC-like physical area directly from FPGA resource utilization estimates (e.g., post-HLS or synthesis reports). MARU achieves an average of 4% Mean Absolute Percentage Error (MAPE) in cross-FPGA experiments using the PolyBench suite, while reducing estimation time from the order of 2.5 days of full ASIC synthesis and place-and-route to just 5 minutes per design. The framework enables three key capabilities: (1) unified area-based comparison across heterogeneous FPGA platforms, (2) real-time area prediction for accelerated HLS design space exploration, and (3) ASIC migration feasibility analysis through technology-aware predictive area modeling. MARU bridges the FPGA–ASIC methodology gap, serving both as a comparative research tool and as a practical framework for cost-driven hardware co-design decisions.

Keywords: Area Estimation · Machine Learning for EDA · FPGA-to-ASIC Mapping · High-Level Synthesis

1 Introduction

Field-Programmable Gate Arrays (FPGAs) have become a dominant platform for rapid prototyping, and accelerating applications in machine learning, communications, and embedded systems [19][2][3]. Their reconfigurable nature enables rapid prototyping, hardware/software co-design, and workload-specific optimization. In both academia and industry, FPGAs are increasingly used as a first step in the hardware design lifecycle—allowing designers to validate architectures, explore microarchitectural trade-offs, and evaluate performance before

committing to fabrication. However, design tradeoff evaluation is complicated by vendor-specific and architecture-specific resource metrics (e.g., Look Up Tables (LUTs) [2][3], adaptive Logic Modules (ALMs) [19], Digital Signal Processing (DSP) blocks) [1][4], which prevent direct cross-platform comparison.

The FPGA design ecosystem reports resource usage using heterogeneous, architecture-specific primitives such as LUTs, FFs, DSPs, BRAMs [2][3][6][35] and ALMs [19]. These primitives differ not only across vendors (e.g., LUTs vs. ALMs) but also across FPGA generations (e.g., DSP48 [1] vs. DSP58 [4] blocks).

This heterogeneity creates two fundamental barriers - First, even within the FPGA domain, fair comparison between designs is challenging. For example, one accelerator implementation may consume more BRAMs but fewer LUTs, while another exhibits the opposite trade-off [35]. Without a common abstraction layer, determining which design is more area-efficient, or more ASIC-ready, is ambiguous. This ambiguity becomes even more pronounced across vendors, where architectural primitives differ substantially in granularity and functionality.

Second, and more critically, there is no direct mapping from FPGA resource utilization to ASIC area. Unlike ASIC flows, where Electronic Design Automation (EDA) tools report area in standardized physical units (e.g., mm^2) derived from technology libraries, FPGA flows lack a unified, technology-aware area metric. As a result, designers cannot easily estimate whether an FPGA prototype will translate into a feasible or cost-effective ASIC implementation. This limits FPGA's effectiveness as a prototyping vehicle for silicon-bound designs.

The challenge is amplified in High-Level Synthesis (HLS)-based flows. HLS [32] is widely adopted for rapid design space exploration, where numerous variants are generated and compared. However, HLS-reported resource estimates are often inaccurate, and obtaining precise post-place-and-route results requires time-consuming compilation flows. When exploring large design spaces, this can require days per design point. Moreover, even when resource estimates are available, whether from HLS or from machine learning-based predictors, the outputs remain disaggregated (LUTs, FFs, DSPs, BRAMs), providing no direct insight into ASIC-equivalent area or migration feasibility.

In this paper, we argue that area can become a unifying abstraction layer for comparison of different FPGA designs and for migrating between FPGA prototyping and ASIC realization. Area is a reasonable proxy for power and cost, making accurate area estimation critical for feasibility. We propose MARU (Machine Learning-based Approaches for Resource-to-Area Estimation), a framework that predicts ASIC-equivalent area directly from FPGA resource utilization statistics. By learning the relationship between FPGA primitives and synthesized ASIC area, MARU enables technology-aware area estimation without requiring full ASIC synthesis and place-and-route.

A naive approach is to estimate design area from resource counts using area weights for each resource, as explored in prior works [7–9]. We applied a similar methodology utilizing area ratios of various FPGA building blocks from VTR's [31] flagship FPGA architecture to estimate an equivalent area. Since the ASIC

area depends on specific technology nodes, we used linear regression to scale it. We ran this experiment for all Polybench designs from HLSDataset [36] - `atax`, `bicg`, `gemm`, `gesummv`, `k2mm`, `k3mm`, `mvt`, `syrk`, and `syr2k`. Across these designs, the MAPE between the area obtained from ASIC synthesis and the area obtained by using area weights from the VTR architecture was 57%, with the worst MAPE being 66.97% for `k3mm` and least MAPE being 46.67% for `k3mm`. The large errors observed here provides a solid motivation for our framework.

To realize our objective, we train machine learning models to estimate the area required for a given FPGA design. We use utilization statistics reported by commercial HLS EDA tools like Vitis [5] as input features, and extract an area estimate from open-source ASIC synthesis tools like Yosys [40] as labels.

The use of ML is critical because the relationship between high-level FPGA resource metrics (such as LUTs, FFs, and DSPs) and area is highly complex and not amenable to analytical modeling or linear regression. This complexity arises from complex architectural patterns—such as variations in DSP block structures or logic cluster configurations—which are difficult to capture analytically, or different circuits having different wiring complexities. For example, across Polybench designs from HLSDataset [36], `gemm` benchmarks exhibit greater area variance than `atax` across different FPGA families, highlighting the need for circuit-architecture-aware prediction. ML-based prediction not only captures these complexities but also offers fast inference times and high accuracy. In this way, our approach offers a fast and generalizable abstraction layer for resource estimation across circuit types and FPGA architectures. Our prediction framework can operate at the HLS or RTL levels.

The proposed MARU framework correlates HLS-derived features with gate-level implementation metrics extracted from both commercial and open-source synthesis flows. MARU is trained on datasets that include multi-FPGA implementation statistics for standard benchmark suites such as PolyBench [36]. It incorporates high-level utilization features (e.g., LUT, FF, DSP, and BRAM counts) and maps them to an area estimate derived using both proprietary and open-source backend flows.

While traditional flows involve running multiple tools across synthesis, place and route, our ML-based MARU flow predicts metrics available at lower design abstraction levels much earlier using reports from higher abstraction levels. Thanks to recent advances in ML, particularly tree-based ensemble models such as XGBoost and Random Forest, MARU captures nonlinear relationships between resource features and area, achieving average percentage errors as low as 1–4% across unseen circuit configurations. While a traditional flow running Vitis for HLS and Vivado/Yosys for synthesis, place and route per circuit requires around 2.5 days to report area on large datasets like HLSDataset [36], the trained MARU model itself requires sub-millisecond inference time per design, with lightweight transfer adaptation completing in under one second on average. As summarized in Table 1, MARU provides key advantages over traditional flows: it offers predictive modeling without re-running the backend flow, supports transfer learning to enable generalization across circuits and FPGAs, and pro-

Table 1. Comparison of traditional FPGA flow and MARU metric and workflow.

Metric	**Traditional Flow**	**MARU Flow (Ours)**
Flow steps	HLS → Synthesis, place and route	HLS → ML inference
Comparison metrics	No	Yes
Dev. cost	None	Requires large dataset
User cost	Vivado or similar tools	(1) Inference only (no effort)
		(2) Transfer learning + inference
Time (user)	Minutes–hours	(1) < 0.12 ms (inference)
		(2) < 1 s (transfer + inference)
Average Error	0%	4%

[1] Model trained on user's circuit/FPGA.
[2] Dataset excludes user's circuit/FPGA.

duces results in real-time during early-stage design. Cross-circuit or cross-FPGA analysis using traditional flows to explore new design spaces typically requires a full rerun of the toolchain. In contrast, MARU allows circuit knowledge to be reused across architectures, enabling efficient estimation through direct prediction or lightweight transfer learning (<10 min), even for incremental design updates. In this work, we explore specific cases based on circuit varieties and cross-platform FPGA varieties.

In this work, we evaluate these capabilities across multiple circuit benchmarks and FPGA platforms to demonstrate scalability and generalization.

Our contributions extend prior research in following key directions:

- We build a systematic machine learning-based framework, MARU, to predict ASIC area from FPGA resource utilization data. MARU achieves accuracy in the range of 94%-99% for circuits from PolyBench from HLSDataset.
- We explore different machine-learning algorithms: linear models like regression, decision trees to complex models like multi-layer perceptrons, neural networks, random forest and gradient boosted decision trees (XGB) accounting for nonlinearities in predictions.
- Intra-circuit and cross-FPGA analysis to test model robustness and generalization capabilities are evaluated as part of MARU. We evaluate direct and transfer-learning protocols to fine-tune predictions.

2 Related Work

The Need for a Single Comparison Metric: Many prior works on FPGA hardware design focus on optimizing metrics like area or performance per unit resource. These are often reported in terms of Configuration Logic Blocks (CLBs) [7, 8] plus DSP slices and BRAMs. Some studies exploring FPGA versus ASIC efficiency gaps [9] convert resource utilization into Adaptive Logic Modules (ALMs), using the number of ALMs to represent area, where various building blocks are converted to an equivalent ALM. However, such vendor-specific

representations limit the generalizability of results across different architectures. This is because the number and structure of logic elements—such as LUTs, flip-flops (FFs), and multiplexers (MUXs)—within a CLB or ALM can vary widely between FPGA vendors and even across different generations from the same vendor. For example, a CLB in a Xilinx and Zynq UltraScale+ FPGA [2, 3] may contain 6-input LUTs, while a Lattice ECP5 and iCE40 FPGA [24, 25] may use 4-input LUTs, and Intel's ALMs follow their own logic structure. A unit ALM contains one LUT and one Flip-Flop. This would correspond to 1/10 of a CLB in Altera FPGAs. A unit DSP and BRAM area corresponds to 40 and 30 ALMs respectively [34]. Such an architecture-specific metric makes it difficult to compare area or performance fairly between designs implemented on different hardware. To solve this, some works report the usage of raw components like LUTs, FFs, BRAMs, and DSPs across designs. For example, [35] evaluated custom FPGA accelerators on neural networks by comparing these components. However, because these metrics depend heavily on the FPGA architecture, toolchain, and synthesis settings, the results are often not transferable or comparable across platforms. This motivates the need for a common comparison metric, such as an area estimate, which can abstract over architecture-specific implementations and enable cross-platform comparisons for area, performance, and resource efficiency.

HLS based Design: High-level synthesis (HLS) has become increasingly widespread in the FPGA design landscape, indicating a significant shift from traditional register-transfer level (RTL) methodologies. As indicated in [10, 32], HLS has moved from a prototyping tool to production with broader accessibility. HLS offers several advantages over RTL-based design, including faster development cycles [23], higher abstraction levels [42], and the ability to leverage software engineering practices, which collectively reduce the time-to-market [18] and lower the barrier for hardware acceleration [22]. HLS facilitates rapid design space exploration and easier integration of complex algorithms, making it more adaptable to evolving application requirements [15, 21]. Furthermore, HLS also enables code optimizations leveraging software engineering practices to simplify hardware acceleration [26].

Machine Learning for Early HLS Metric Prediction: High-Level Synthesis (HLS), which converts C/C++ code into RTL, is widely used in modern FPGA workflows [10, 17, 18, 21–23, 42]. However, the process of evaluating tradeoffs—like resource usage, power, and timing—is slow, since full synthesis and implementation tools can take hours to days per design iteration. This slows down rapid exploration of design alternatives. To address this, many recent works use Machine Learning (ML) models to predict post-synthesis metrics such as LUT counts, FF counts, and critical path delays. Techniques like convolutional neural networks (CNNs), graph neural networks (GNNs), and tree-based models (e.g., XGBoost) have been applied to predict key HLS metrics, including resource usage [12], power consumption [27, 28, 37], and timing performance [41]. Dai et al. [13], for example, showed that using synthesis report features can reduce HLS estimation errors by up to 138%. Similarly, Wu et al. [41] used GNNs on abstract syntax trees to predict post-implementation timing. Large datasets like HLSDataset [36], which includes over 9,000 design variants across

multiple FPGA families, have enabled these models to be trained effectively at scale. Still, challenges remain in making predictions that generalize well across different FPGAs, toolchains, and designs.

Cross-Platform Prediction and Transfer Learning: Recent works have explored the challenge of predicting design metrics across different FPGA platforms [29, 33, 38, 39]. ATAPP [38] and XPNet [39] predict the power consumption on unseen FPGAs using machine learning. HLSPredict [33] applies ML models to predict performance for new FPGA targets without running the full toolchain. Similarly, XPPE [29] estimates kernel-level performance for unseen hardware types using CPU execution traces. These methods aim to reduce the time and compute resources needed for evaluating new designs or migrating designs across devices. In the context of HLS, machine learning models trained on one FPGA architecture often perform poorly when applied to another, due to differences in logic block configurations, synthesis behavior, and implementation tools. To address this, transfer learning has emerged as a powerful method. Transfer learning allows a model trained on one dataset to be adapted to a new target using a small amount of labeled data—greatly reducing the need for full retraining and large-scale data collection. For HLS-based flows, where generating post-implementation metrics like area, power, and timing typically requires a full synthesis and place-and-route run, transfer learning enables fast reuse of existing knowledge. This makes it particularly useful for design space exploration (DSE) under resource constraints. Previous works such as [14, 20] demonstrate that transfer learning can speed up DSE across different circuits or FPGAs, reducing latency while maintaining high prediction accuracy.

3 Methodology

The data set used in MARU is based on a HLS dataset [36] that could be used for rapid development of machine learning platforms with end-to-end integration. HLS enables the translation of high-level descriptions (e.g., C/C++) into hardware implementations while exposing design trade-offs through resource utilization reports [11]. It consists of 9 different benchmarks, each containing atleast 500 different configurations parametrized by applying pragmas to each kernel in the Polybench benchmark suites - array partitioning, loop unrolling, and pipelining factors. This results in a net **total of 4,500+ circuits available for training and inference**. Since the HLS dataset contains proprietary Xilinx modules for implementing optimized floating point addition and multiplication operations, custom modules needed to be added to replicate this in ASIC gate-level synthesis.

3.1 MARU data generation framework

The flow for data generation is as shown in Figure 1. The Polybench benchmark consists of C++ benchmark with pragma statements for loop unrolling, array partitioning and pipelining. These are used to run HLS in Xilinx Vitis FPGA to generate Verilog files and utilization reports. The utilization reports contain

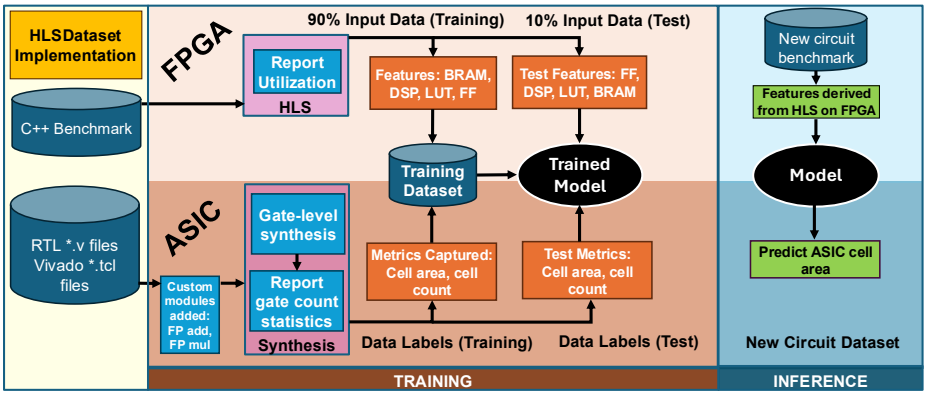

Fig. 1. MARU training and inference frameworks using Vitis HLS for FPGA and Yosys for ASIC gate-level synthesis implementations.

features such as BRAMs, DSPs, FFs, and LUT counts used in FPGA design. We use Vitis HLS [5] to generate these HLS resource utilization reports.

For building output data label set containing standard cell area estimates for an equivalent ASIC implementation using Yosys and which are to be predicted by a machine-learning model, we process HLS Verilog files containing circuit implementations and any other Vivado TCL files as input to the ML model. Since the design from HLSDataset contains binary files with Xilinx proprietary modules, we designed custom modules like floating-point addition and multiplication. ASIC gate-level synthesis is done in Yosys [40] using a basic synthesis recipe with no tool specific parametric optimizations. FPGA resource utilization from HLSDataset and ASIC post-implementation area summary reports from Yosys are parsed and merged to create a single comma-separated values (CSV) file with training data and labels. Model training is done using training and test datasets. Trained model is validated using inference dataset.

Table 2 shows a snippet of FPGA input data and generated equivalent ASIC area per benchmark. Here area is the output that needs to be predicted. Along with total cell area and cell count, Yosys also provides cell counts for different types of cells used in ASIC design. We use Skywater130 PDK [16] for implementation. During implementation, unit area per cell for a given technology is known. Thus, estimating cell area and cell count per cell-type provides an accurate picture of total chip/silicon area needed for implementing a given circuit in ASIC. At an abstract-level we do see a correlation between LUT, FF and DSP counts with total cell area. Further, the number of registers/ flip-flops and block RAMs influences the various D-type flip-flop counts after gate-level synthesis.

3.2 MARU Learning Framework

In this section, we describe a systematic machine learning-based framework, MARU to evaluate various ML models across multiple performance metrics, including R^2, Root Mean-Square Error (RMSE), and MAPE. The learning pipeline begins with simple models such as linear regression and multi-layer perceptrons,

Table 2. Training data summary. Input features from Vitis HLS, while area labels are extracted from Yosys gate-level synthesis.

Benchmark	LUT	FF	DSP	BRAM	CellArea (μm^2)
ATAX	1054	680	5	11	2,396,601
BICG	1103	781	13	5	2,466,626
GEMM	4091	4649	32	5	12,106,403
GESUMMV	1212	739	20	5	4,677,871
K2MM	3827	2505	56	10	18,734,007
K3MM	2096	1164	56	5	15,763,217
MVT	1074	679	12	5	2,432,109
SYRK	1412	829	32	5	9,025,421
SYR2K	5069	3276	56	14	19,554,186

and progressively explores more expressive models like neural networks and ensemble learners (e.g., Random Forest and XGBoost) to capture underlying nonlinear relationships.

A part of this motivation stems from early observations that raw utilization metrics alone do not map linearly to estimated area. For example, although the overall resource usage (LUTs, FFs, DSPs, BRAMs) for benchmarks such as `atax` and `syrk` falls within a similar range, the synthesized area estimate differs significantly. As shown in Table 2, `syrk` uses only moderately more total resources than `atax`, but exhibits nearly 4× higher cell area, largely due to increased DSP utilization. This highlights that differences in resource composition—not just total counts—can lead to substantial variation in resulting area.

Table 3. List of features used for gate count prediction.

Feature Name	Description / Formula
LUT	Number of Look-Up Tables used
FF	Number of Flip-Flops used
DSP	Number of DSP slices used
BRAM	Number of Block RAMs used
LUT_FF_ratio	LUT ÷ (FF + 1)
total_resources	LUT + FF + DSP + BRAM
DSP_LUT_ratio	DSP ÷ (LUT + 1)
log_LUT	$\log(1 + \text{LUT})$
log_FF	$\log(1 + \text{FF})$

To address these discrepancies while preserving generalization, we deliberately kept feature engineering minimal and interpretable in early experiments. The goal was to isolate model performance without relying on complex or highly domain-specific transformations. Only a small number of features—primarily the raw utilization metrics—were used initially, and later experiments introduced

modest extensions such as resource ratios (e.g., LUT/FF), log transformations, and simple aggregations. Table 3 lists the complete set of nine features used in the final models, including both raw inputs and engineered variants. These were selected for their simplicity, interpretability, and relevance across FPGAs.

Feature scaling (via MinMax or RobustScaler) and log transformations (on target values) were applied where needed to normalize feature ranges and improve regression stability. The training data was used to fit four representative models: Multi-Layer Perceptron (MLP), Keras-based neural network, Random Forest, and XGBoost. These models were evaluated using R^2, RMSE, and MAPE metrics, and the best model was chosen based on highest R^2 and lowest error. This is shown in Figure 2.

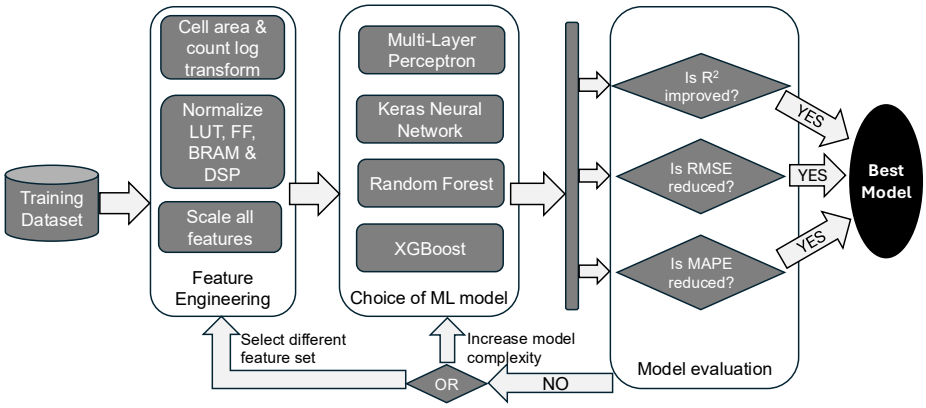

Fig. 2. Feature engineering, model choice and evaluation metrics used in MARU learning framework.

Figure 3 shows implementing an ensemble based machine learning model (e.g., Random Forest) as part of our framework. The input data and data labels that needs to be predicted is merged together to form a training dataset. We apply bootstrap sampling and K-Fold cross validation [30] methods to help reduce variance, add more robustness to predicted models, avoid over-fitting, explore more diverse feature combinations compared to naive weighted co-efficient based predictions in linear regression.

To support cross-FPGA generalization, MARU incorporates transfer learning through fine-tuning and residual-based adaptation. Fine-tuning is performed by warm-starting the pretrained XGBoost model and continuing training on a small subset of labeled designs from the target FPGA (e.g., tens to a few hundred samples), rather than retraining from scratch. In residual-based adaptation, the pretrained model is kept fixed and a lightweight model is trained to predict residual errors on the target domain. Fine-tuning is required only when adapting to a new FPGA family and does not require re-running the synthesis flow for

each new design. After this one-time step, inference on new designs requires only FPGA resource utilization features and incurs sub-millisecond latency, with no additional synthesis or place-and-route steps.

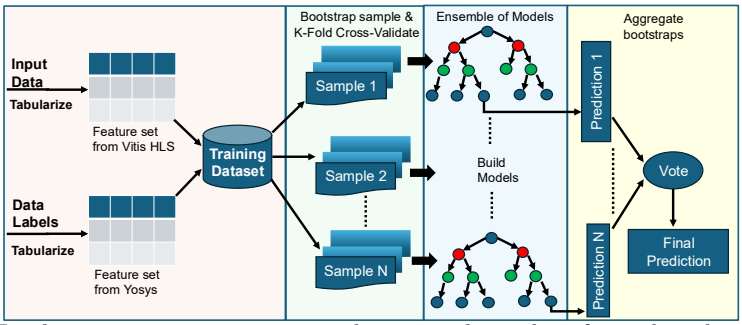

Fig. 3. Implementing area estimate prediction with random forest based predictors, bootstrap sampling and K-Fold cross-validation.

3.3 Experimental Setup Overview

To systematically evaluate the predictive capabilities of the MARU framework, we designed five distinct experiments that vary along benchmarks and prediction settings. Each experiment explores a different generalization axis—ranging from evaluation within the same benchmark and FPGA to multiple benchmarks modeling across different FPGA versions—and informs the design of the training and test splits.

Table 4 summarizes the experimental setup. It outlines the specific scenario explored in each experiment, the number of benchmarks involved, and the nature of the modeling case (e.g., direct prediction vs. transfer learning). This structured evaluation enables a comprehensive assessment of model robustness and portability across diverse circuit configurations.

Circuits are synthesized in FPGA using HLS and ASIC using Yosys with a fixed 10ns clock period. This enables timing-aware resource utilization across both tools. We also used liberty files from open-source Skywater130 PDK [16] to capture standard area estimates.

4 Results

In this section, we evaluate MARU for ASIC area prediction under progressively more challenging generalization settings: (i) across benchmarks and (ii) across FPGA architectures as in Table 4. Performance is reported using the coefficient of determination (R^2), RMSE, and MAPE.

4.1 Experiment 1: Intra-Benchmark Prediction

This experiment corresponds to setting #1 (Same FPGA, Same Benchmark) in Table 4. We establish a baseline by training models on variants of a single benchmark, `atax`, synthesized for the Xilinx `xc7v` FPGA, and evaluating on unseen variants from the same benchmark. The goal is to identify effective

Table 4. Summary of experimental scenarios and cases for training and inference for each experiment and their corresponding modeling configurations.

Exp	Training Benchmarks	Inference Benchmarks	Methods evaluated
1 (Sec 4.1)	FPGA V (80% of A)	FPGA V (20% of A)	Baseline
2 (Sec 4.2)	FPGA V (80% of A)	FPGA V (B, G, M, K, S)	Direct, Fine-tuned
3 (Sec 4.3)	FPGA V (each benchmark 80%)	FPGA Z (same benchmark 20%)	Direct, Fine-tuned
4 (Sec 4.4)	FPGA V (8 benchmarks)	FPGA V (1 benchmark)	Leave-one-out, Fine-tuned
5 (Sec 4.5)	FPGA V (9 benchmarks)	FPGA Z (1 benchmark)	Direct, Fine-tuned

Benchmarks: A=atax, B=bicg, G=gemm/gesummv, M=mvt, K=k2mm/k3mm, S=syrk/syr2k. FPGA: V=xc7v585tffg1157-3, Z=xczu9eg-ffvb1156-2.

model classes for area estimation, and to assess whether accurate prediction is achievable within a relatively homogeneous setting before extending to cross-benchmark and cross-FPGA scenarios.

Linear models fail to capture the underlying complexity of the data due to nonlinear interactions between resource features (e.g., LUT, FF, BRAM). As a result, we evaluate nonlinear models, including tree-based ensemble methods (Random Forest, XGBoost) and neural networks (MLP, Keras). Among these, XGBoost achieves the best performance, and is therefore used in all experiments.

Table 5. Comparison of models for ASIC area prediction on unseen circuit variants.

ML Model	MLP	Keras NN	Random Forest	XGBoost
R^2	-11.42	0.0033	0.9912	0.9942
RMSE (%)	104.30	29.54	2.77	2.26
MAPE (%)	99.99	25.05	1.44	1.47

As shown in Table 5, XGBoost achieves the highest accuracy ($R^2 = 0.9942$, RMSE = 2.26%, MAPE = 1.47%), closely followed by Random Forest ($R^2 = 0.9912$, RMSE = 2.77%, MAPE = 1.44%). In contrast, neural networks perform poorly, with unstable or near-zero R^2 values and significantly higher errors.

4.2 Experiment 2: Inter-Benchmark Prediction

This experiment corresponds to setting #2 (same FPGA, different benchmark) in Table 4. We evaluate how the baseline model trained on `atax` benchmark generalizes to unseen benchmarks (e.g., `bicg`, `gemm`) on the same FPGA. We consider direct prediction, where the trained model is applied to unseen benchmarks without modification, and transfer learning, where the model is fine-tuned using 20% of the target benchmark and evaluated on the remaining 80%.

Table 6. Cross-benchmark transfer from a model trained on `atax`.

	R^2		RMSE		MAPE (%)	
Benchmark	Direct	Transfer	Direct	Transfer	Direct	Transfer
BICG	0.47	0.99	1.36e6	1.32e5	4.49	1.77
GEMM	-10.99	0.93	8.15e6	6.04e5	71.13	3.09
MVT	0.32	0.99	2.13e6	2.59e5	11.22	4.45
GESUMMV	-0.38	1.00	4.24e6	9.14e4	41.48	0.80
K2MM	-23.15	0.89	1.54e7	1.03e6	81.70	3.44
K3MM	-44.55	0.78	1.45e7	1.01e6	82.58	3.34
SYR2K	-17.44	0.94	1.55e7	8.59e5	81.30	2.39
SYRK	-9.43	0.95	7.88e6	5.64e5	70.43	2.53
Average	**-13.52**	**0.94**	**8.28e6**	**6.63e5**	**45.54**	**2.73**

Case 1: Direct Prediction As shown in Table 6, direct prediction performs inconsistently across benchmarks. While simpler benchmarks such as `bicg` achieve moderate accuracy ($R^2 = 0.465$), most benchmarks exhibit poor generalization, with several cases yielding large negative R^2 values (e.g., `k3mm`: –44.55). This indicates that a model trained on a single benchmark does not capture cross-benchmark variability in resource–area relationships.

Case 2: Transfer Learning Applying transfer learning (via fine-tuning on a small subset of target-benchmark data) significantly improves performance across all benchmarks. For example, R^2 improves from –10.99 to 0.934 for `gemm`, from –23.15 to 0.893 for `k2mm`, and from –44.55 to 0.780 for `k3mm`. Even benchmarks with moderate baseline performance (e.g., `bicg`, `mvt`) improve to near-perfect accuracy ($R^2 > 0.99$). As shown in Figure 4, predictions after transfer learning align closely with the ideal diagonal. These results demonstrate that while direct prediction provides limited generalization across benchmarks, lightweight transfer learning enables accurate and robust area estimation.

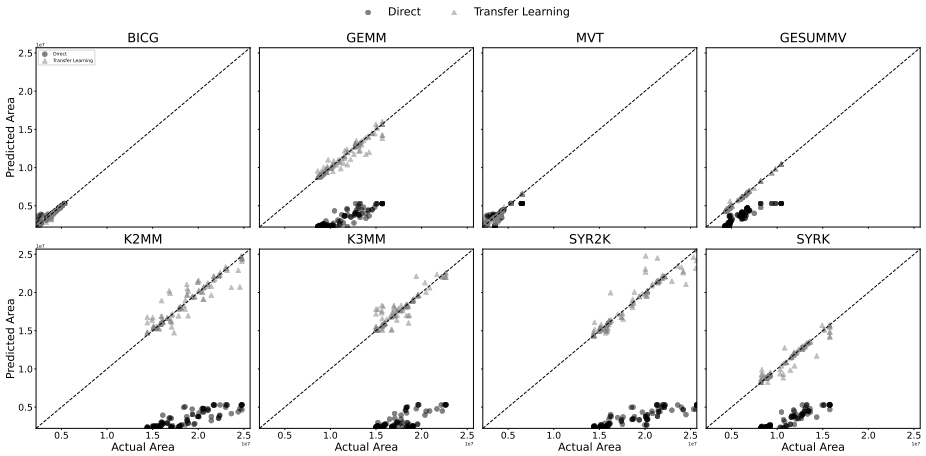

Fig. 4. Transfer learning and prediction of `atax` ML model on different benchmarks. Circles are direct predictions and triangles are transfer learning.

4.3 Experiment 3: Testing Cross-FPGA Generalization

A frequent question is, if a model is trained using a particular FPGA, will the model work even if features are collected on a different FPGA. This experiment corresponds to the setting #3 (same benchmark, different FPGA) in Table 4. We evaluate how a model trained using the `xc7v` FPGA generalizes even if the inference features are obtained from a different FPGA (`xczu9`), while still predicting ASIC-equivalent area metric from FPGA-derived features. We consider direct prediction across FPGA architectures, and residual-based transfer learning, where the model is lightly adapted using target-FPGA data.

Table 7. Cross-FPGA prediction accuracy across Polybench (Experiment 3).

	R^2		RMSE		MAPE (%)	
Benchmark	Direct	Transfer	Direct	Transfer	Direct	Transfer
ATAX	1.0000	1.0000	10120	9633	0.20%	0.21%
BICG	0.9991	0.9992	58685	55556	0.82%	0.82%
GEMM	0.9618	0.9635	452905	442636	2.24%	2.34%
MVT	0.9878	0.9881	241168	237592	2.82%	3.10%
GESUMMV	0.9679	0.9720	652365	608872	1.21%	0.83%
K2MM	0.9682	0.9738	583385	529027	1.57%	1.49%
K3MM	0.9494	0.9560	581696	542479	1.74%	1.86%
SYR2K	0.9869	0.9901	405690	353009	1.04%	1.08%
SYRK	0.9916	0.9921	228287	221292	0.68%	0.72%
Average	**0.9792**	**0.9816**	**357144**	**333344**	**1.37**	**1.38**

Case 1: Direct Prediction As shown in Table 7, direct cross-FPGA prediction achieves strong baseline performance, with an average R^2 of 0.979, indicating that the learned resource–area relationships transfer well across FPGA architectures. This is further supported by low prediction error, with an average RMSE of 357,144 and MAPE of 1.37%, demonstrating accurate absolute and relative estimation across benchmarks.

Case 2: Transfer Learning Applying residual-based transfer learning further improves performance, increasing the average R^2 to 0.9816 while reducing RMSE to 333,344 and maintaining low MAPE (1.38%). Direct prediction already produces close estimates, but residual-based transfer learning reduces outliers and improves consistency. Across all benchmarks, transfer learning yields consistent improvements (average $\Delta R^2 \approx +0.0043$), with larger gains observed in compute-intensive benchmarks such as `k3mm` (+0.0066) and `k2mm` (+0.0057), where FPGA-specific DSP and BRAM mappings differ more significantly. These results indicate that cross-FPGA generalization is inherently strong, and lightweight transfer learning enhances robustness.

4.4 Experiment 4: Multi-Benchmark Prediction

This experiment corresponds to setting #4 (different benchmark, same FPGA) in Table 4. We train a unified model across multiple benchmarks and evaluate generalization using a leave-one-benchmark-out strategy. In each iteration, one benchmark is held out for testing, while the model is trained on the remaining

benchmarks. We then apply transfer learning by fine-tuning on 50% of the held-out benchmark.

Case 1: Direct Prediction As shown in Table 8, direct prediction gives mixed results across benchmarks. Memory-bound benchmarks such as `bicg` (R^2 = 0.795) and `mvt` (R^2 = 0.831) achieve reasonable accuracy, showing that learned patterns can be partially transferred. In contrast, compute-intensive designs such as `k2mm` (R^2 = –0.805) and `k3mm` (R^2 = –3.205) perform poorly, reflecting differences in DSP/BRAM-driven scaling behavior. This variability is also reflected in the error metrics, with a high average RMSE of 2,608,956 and MAPE of 20.15%, indicating poor generalization to unseen resource distributions. Overall, direct multi-benchmark prediction is inconsistent, with several benchmarks exhibiting negative R^2 values.

Table 8. Multi-benchmark leave-one-out validation results (Experiment 4).

	R^2		RMSE		MAPE (%)	
Benchmark	Direct	Transfer	Direct	Transfer	Direct	Transfer
ATAX	0.42	0.99	1532263	217783	24.20%	1.28%
BICG	0.80	1.00	909064	128045	11.94%	1.19%
GEMM	0.21	0.95	2073505	538106	11.15%	2.30%
GESUMMV	-1.00	1.00	5075980	210338	67.98%	0.76%
K2MM	-0.80	0.94	4286101	779003	13.24%	1.90%
K3MM	-3.21	0.93	4942444	635418	16.85%	1.58%
MVT	0.83	0.99	1066137	256862	16.59%	3.22%
SYR2K	-0.12	0.95	3889193	857838	13.98%	1.74%
SYRK	0.67	0.97	1405916	448094	7.41%	1.37%
Average	**-0.24**	**0.97**	**2608956**	**452832**	**20.15**	**1.71**

Case 2: Transfer Learning Applying transfer learning significantly improves performance across all benchmarks. As shown in Table 8, the largest gains occur for benchmarks with poor baseline performance—for example, `gesummv` improves from –0.997 to 0.997, and `k3mm` from –3.205 to 0.930. Even benchmarks with moderate baseline accuracy (e.g., `bicg`, `mvt`) improve performance ($R^2 > 0.99$).

In addition to the R^2 improvement, transfer learning substantially reduces prediction error, lowering the average RMSE from 2,608,956 to 452,832 and MAPE from 20.15% to 1.71%. The results in Table 8 show that while direct prediction is unreliable (mean R^2 = –0.2449), transfer learning produces consistently accurate estimates (mean R^2 = 0.97). A multi-benchmark model provides a strong starting point, but accurate area estimation across different benchmarks requires lightweight transfer learning to adapt to each benchmark.

4.5 Experiment 5: Multi-Benchmark Cross-FPGA Generalization

This experiment evaluates whether a multi-benchmark model trained on `xc7v` data can generalize to `xczu9` while still predicting a unified ASIC-equivalent area metric from FPGA-derived features, and corresponds to setting #5 in Table 4. We report both direct cross-FPGA prediction and residual-based transfer learning for adaptation.

Case 1: Direct Prediction Table 9 summarizes direct prediction results using a multi-benchmark model trained on `xc7v`. Direct transfer achieves a strong baseline (average $R^2 = 0.8948$), but performance varies across benchmarks, with the largest degradation observed for compute-intensive benchmarks such as `k3mm`. This variability is also reflected in the error metrics, with an average RMSE of 2.82M and MAPE of 4.96%, indicating that cross-FPGA transfer becomes more challenging when training across diverse benchmarks.

Table 9. Multi-benchmark model Cross-FPGA prediction accuracy (Experiment 5).

	R^2		RMSE		MAPE (%)	
Benchmark	Direct	Transfer	Direct	Transfer	Direct	Transfer
ATAX	0.99	0.99	0.15M	0.14M	2.63	2.73
BICG	0.99	0.99	0.17M	0.14M	3.47	3.11
GEMM	0.84	0.88	0.92M	0.81M	4.64	4.20
MVT	0.96	0.98	0.42M	0.31M	7.28	5.85
GESUMMV	0.92	0.96	1.05M	0.72M	10.89	5.60
K2MM	0.89	0.92	1.11M	0.95M	3.78	3.31
K3MM	0.66	0.78	1.51M	1.20M	4.65	4.20
SYR2K	0.87	0.88	1.28M	1.23M	3.92	4.16
SYRK	0.93	0.94	0.64M	0.60M	3.34	2.95
Average	**0.89**	**0.93**	**2.82M**	**1.08M**	**4.96**	**4.01**

Case 2: Residual-Based Transfer Learning Residual-based transfer learning (adapted using 20% of the target `xczu9` data) improves robustness and increases average performance to $R^2 = 0.9252$ (Table 9). In addition to the R^2 improvement, transfer learning reduces prediction error, lowering the average RMSE from 2.82M to 1.08M and MAPE from 4.96% to 4.01%. Gains are most pronounced for benchmarks that exhibit larger cross-FPGA mismatch under direct prediction (e.g., `k3mm` and `gesummv`), indicating that residual adaptation helps correct FPGA-specific implementation differences. Overall, these results show that a unified multi-benchmark model transfers effectively across FPGA generations, and lightweight residual adaptation further improves consistency and reduces large errors, particularly for benchmarks with distinct DSP/BRAM-driven scaling behavior.

5 Conclusion

In this work, we built a learning-based framework, MARU, to predict ASIC area estimates from Vitis HLS resource utilization statistics. Across multiple carefully designed experiments, the MARU framework demonstrated strong predictive performance and generalization across circuits and FPGAs. MARU achieves an average MAPE of around 4% in cross-FPGA, multi-benchmark scenarios, with R^2 values up to 0.99. While per-benchmark models achieve near-perfect accuracy, generalization across benchmarks and platforms benefits significantly from transfer learning. Inference requires less than 0.12 ms per design on average, while lightweight transfer adaptation completes in under one second.

Acknowledgments. This research was supported in part by NSF Grants 2326894, 2425655, and 2417658. Any opinions, findings, conclusions, or recommendations are those of the authors and not of the funding agencies.

Disclosure of Interests. Authors have no competing interests.

References

1. AMD Xilinx: 7 series dsp48e1 slice user guide (ug479) (2018), https://www.xilinx.com/support/documentation/user_guides/ug479_7Series_DSP48E1.pdf
2. AMD Xilinx: 7 series fpgas data sheet: Overview (virtex-7, kintex-7, artix-7) (2018), https://www.xilinx.com/support/documentation/data_sheets/ds180_7Series_Overview.pdf
3. AMD Xilinx: Ultrascale architecture and product data sheet (2019), https://www.xilinx.com/support/documentation/data_sheets/ds890-ultrascale-overview.pdf
4. AMD Xilinx: Versal dsp58 architecture (2023), https://docs.amd.com/r/en-US/am004-versal-dsp-engine/DSP58-Architecture
5. AMD Xilinx: Vitis high-level synthesis (hls) (2023), https://www.xilinx.com/products/design-tools/vitis/vitis-hls.html
6. Bacellar, A.T.L., Susskind, Z., Breternitz Jr, M., John, E., John, L.K., Lima, P.M.V., França, F.M.: Differentiable weightless neural networks. In: Salakhutdinov, R., Kolter, Z., Heller, K., Weller, A., Oliver, N., Scarlett, J., Berkenkamp, F. (eds.) Proceedings of the 41st International Conference on Machine Learning. Proceedings of Machine Learning Research, vol. 235, pp. 2277–2295. PMLR (21–27 Jul 2024), https://proceedings.mlr.press/v235/bacellar24a.html
7. Boutros, A., Nurvitadhi, E., Betz, V.: Specializing for efficiency: Customizing ai inference processors on fpgas. In: 2021 International Conference on Microelectronics (ICM). pp. 62–65 (2021). https://doi.org/10.1109/ICM52667.2021.9664938
8. Boutros, A., Yazdanshenas, S., Betz, V.: Embracing diversity: Enhanced dsp blocks for low-precision deep learning on fpgas. In: 2018 28th International Conference on Field Programmable Logic and Applications (FPL). pp. 35–357 (2018). https://doi.org/10.1109/FPL.2018.00014
9. Boutros, A., Yazdanshenas, S., Betz, V.: You cannot improve what you do not measure: Fpga vs. asic efficiency gaps for convolutional neural network inference **11**(3) (2018), https://doi.org/10.1145/3242898
10. Cong, J., Lau, J., Liu, G., Neuendorffer, S., Pan, P., Vissers, K., Zhang, Z.: Fpga hls today: Successes, challenges, and opportunities. ACM Transactions on Reconfigurable Technology and Systems **15**(4), 1–42 (2022)
11. Cong, J., Liu, B., Neuendorffer, S., Noguera, J., Vissers, K., Zhang, Z.: High-level synthesis for fpgas: From prototyping to deployment. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems **30**(4), 473–491 (2011). https://doi.org/10.1109/TCAD.2011.2110592
12. Dai, S., Zhou, Y., Zhang, H., Ustun, E., Young, E.F., Zhang, Z.: Fast and accurate estimation of quality of results in high-level synthesis with machine learning. In: 2018 IEEE 26th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM). pp. 129–132 (2018). https://doi.org/10.1109/FCCM.2018.00029

13. Dai, S., Zhou, Y., Zhang, H., Ustun, E., Young, E.F., Zhang, Z.: Fast and accurate estimation of quality of results in high-level synthesis with machine learning. In: 2018 IEEE 26th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM). pp. 129–132 (2018). https://doi.org/10.1109/FCCM.2018.00029
14. Ding, Z., Sohrabizadeh, A., Li, W., Qin, Z., Sun, Y., Cong, J.: Efficient Task Transfer for HLS DSE. Association for Computing Machinery, New York, NY, USA (2025), https://doi.org/10.1145/3676536.3676723
15. Du, Y., Hu, Y., Zhou, Z., Zhang, Z.: High-performance sparse linear algebra on hbm-equipped fpgas using hls: A case study on spmv. Association for Computing Machinery, New York, NY, USA (2022), https://doi.org/10.1145/3490422.3502368
16. Google, SkyWater Technology Foundry: SkyWater Open Source PDK. https://github.com/google/skywater-pdk (2020), accessed: 2025-09-30
17. Goswami, P., Shahshahani, M., Bhatia, D.: Mlsbench: A benchmark set for machine learning based fpga hls design flows. In: 2022 IEEE 13th Latin America Symposium on Circuits and System (LASCAS). pp. 1–4 (2022). https://doi.org/10.1109/LASCAS53948.2022.9789084
18. Guo, L., Maidee, P., Zhou, Y., Lavin, C., Wang, J., Chi, Y., Qiao, W., Kaviani, A., Zhang, Z., Cong, J.: Rapidstream: Parallel physical implementation of fpga hls designs. Association for Computing Machinery, New York, NY, USA (2022), https://doi.org/10.1145/3490422.3502361
19. Intel: Stratix v device handbook (2018), https://www.intel.com/content/www/us/en/docs/programmable/683665
20. Kwon, J., Carloni, L.P.: Transfer learning for design-space exploration with high-level synthesis. In: 2020 ACM/IEEE 2nd Workshop on Machine Learning for CAD (MLCAD). pp. 163–168 (2020). https://doi.org/10.1145/3380446.3430636
21. Lahti, S., Hämäläinen, T.D.: High-level synthesis for fpgas—a hardware engineer's perspective. IEEE Access **13**, 28574–28593 (2025). https://doi.org/10.1109/ACCESS.2025.3540320
22. Lahti, S., Sjövall, P., Vanne, J., Hämäläinen, T.D.: Are we there yet? a study on the state of high-level synthesis. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems **38**(5), 898–911 (2019). https://doi.org/10.1109/TCAD.2018.2834439
23. Lai, Y.H., Rong, H., Zheng, S., Zhang, W., Cui, X., Jia, Y., Wang, J., Sullivan, B., Zhang, Z., Liang, Y., Zhang, Y., Cong, J., George, N., Alvarez, J., Hughes, C., Dubey, P.: Susy: A programming model for productive construction of high-performance systolic arrays on fpgas. In: 2020 IEEE/ACM International Conference On Computer Aided Design (ICCAD). pp. 1–9 (2020)
24. Lattice Semiconductor: ice40 ultraplus family data sheet (2019), https://www.latticesemi.com/-/media/LatticeSemi/Documents/DataSheets/iCE/iCE40-UltraPlus-Family-Data-Sheet.pdf
25. Lattice Semiconductor: Lattice ecp5 family data sheet (2020), https://www.latticesemi.com/-/media/LatticeSemi/Documents/DataSheets/ECP-5/FPGA-DS-02012.pdf
26. Lau, J., Sivaraman, A., Zhang, Q., Gulzar, M.A., Cong, J., Kim, M.: Heterorefactor: refactoring for heterogeneous computing with fpga. Association for Computing Machinery, New York, NY, USA (2020), https://doi.org/10.1145/3377811.3380340
27. Lin, Z., Yuan, Z., Zhao, J., Zhang, W., Wang, H., Tian, Y.: Powergear: Early-stage power estimation in FPGA HLS via heterogeneous edge-centric gnns. CoRR (2022)

28. Lin, Z., Zhao, J., Sinha, S., Zhang, W.: Hl-pow: A learning-based power modeling framework for high-level synthesis. In: 2020 25th Asia and South Pacific Design Automation Conference (ASP-DAC). pp. 574–580 (2020). https://doi.org/10.1109/ASP-DAC47756.2020.9045442
29. Makrani, H.M., Sayadi, H., Mohsenin, T., rafatirad, S., Sasan, A., Homayoun, H.: Xppe: cross-platform performance estimation of hardware accelerators using machine learning. Association for Computing Machinery, New York, NY, USA (2019), https://doi.org/10.1145/3287624.3288756
30. Marcot, B., Hanea, A.: What is an optimal value of k in k-fold cross-validation in discrete bayesian network analysis? Computational Statistics **36** (09 2021). https://doi.org/10.1007/s00180-020-00999-9
31. Murray, K.E., Petelin, O., Zhong, S., Wang, J.M., Eldafrawy, M., Legault, J.P., Sha, E., Graham, A.G., Wu, J., Walker, M.J.P., Zeng, H., Patros, P., Luu, J., Kent, K.B., Betz, V.: Vtr 8: High-performance cad and customizable fpga architecture modelling **13**(2) (2020), https://doi-org.ezproxy1.lib.asu.edu/10.1145/3388617
32. Nane, R., Sima, V.M., Pilato, C., Choi, J., Fort, B., Canis, A., Chen, Y.T., Hsiao, H., Brown, S., Ferrandi, F., Anderson, J., Bertels, K.: A survey and evaluation of fpga high-level synthesis tools **35**(10) (2016), https://doi.org/10.1109/TCAD.2015.2513673
33. O'Neal, K., Liu, M., Tang, H., Kalantar, A., DeRenard, K., Brisk, P.: Hlspredict: Cross platform performance prediction for fpga high-level synthesis. In: 2018 IEEE/ACM International Conference on Computer-Aided Design (ICCAD). pp. 1–8 (2018). https://doi.org/10.1145/3240765.3240816
34. Rashid, R., Steffan, J.G., Betz, V.: Comparing performance, productivity and scalability of the tilt overlay processor to opencl hls. In: 2014 International Conference on Field-Programmable Technology (FPT). pp. 20–27 (2014). https://doi.org/10.1109/FPT.2014.7082748
35. Susskind, Z., Arora, A., Miranda, I.D.S., Villon, L.A.Q., Katopodis, R.F., de Araújo, L.S., Dutra, D.L.C., Lima, P.M.V., França, F.M.G., Breternitz, M., John, L.K.: Weightless neural networks for efficient edge inference. Association for Computing Machinery, New York, NY, USA (2023), https://doi.org/10.1145/3559009.3569680
36. Wei, Z., Arora, A., Li, R., John, L.: Hlsdataset: Open-source dataset for ml-assisted fpga design using high level synthesis. In: 2023 IEEE 34th International Conference on Application-specific Systems, Architectures and Processors (ASAP). pp. 197–204 (2023). https://doi.org/10.1109/ASAP57973.2023.00040
37. Wei, Z., Arora, A., Shriver, E., John, L.: Cross-fpga power estimation from high level synthesis via transfer-learning. Association for Computing Machinery, New York, NY, USA (2024), https://doi.org/10.1145/3626202.3637621
38. Wei, Z., Arora, A., Shriver, E., John, L.: Atapp: Architecture and technology aware power predictor for unseen fpgas (2025)
39. Wei, Z., Seigler, A., Lowe, S., Shriver, E., Arora, A., John, L.K.: Xpnet: Cross-fpga power prediction from high level language code. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (2025)
40. Wolf, C.: Yosys open synthesis suite. https://yosyshq.net/yosys/
41. Wu, N., Yang, H., Xie, Y., Li, P., Hao, C.: High-level synthesis performance prediction using gnns: benchmarking, modeling, and advancing. Association for Computing Machinery, New York, NY, USA (2022), https://doi.org/10.1145/3489517.3530408

42. Xiang, S., Lai, Y.H., Zhou, Y., Chen, H., Zhang, N., Pal, D., Zhang, Z.: Heteroflow: An accelerator programming model with decoupled data placement for software-defined fpgas. Association for Computing Machinery, New York, NY, USA (2022), https://doi.org/10.1145/3490422.3502369