This article has been accepted for publication in IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems. This is the author's version which has not been fully edi
content may change prior to final publication. Citation information: DOI 10.1109/TCAD.2025.3646555

XPNet: Cross-FPGA Power Prediction from High Level Language Code

Zhigang Wei*, Allison Seigler*, Sean Lowe†, Emily Shriver, Aman Arora† and Lizy K. John *
*Department of Electrical and Computer Engineering, The University of Texas at Austin
†School of Computing and Augmented Intelligence, Arizona State University

***Abstract*—Machine learning (ML) has been successfully employed to estimate power consumption for FPGAs using features derived from the results of High Level Synthesis (HLS). However, such models trained on one FPGA cannot be directly applied to another FPGA, even within the same FPGA series. Training a model for a new FPGA is time-consuming due to the significant effort required for dataset preparation. Researchers have to invest significant effort (weeks) in constructing a sufficient dataset with power value annotations, to train an accurate model for a new target FPGA. Another challenge is that existing model construction methods depend on many features extracted late in the HLS process, which are tool-specific and cannot be transferred between tools from different vendors.**

To address these challenges, we propose a novel cross-FPGA power modeling methodology called XPNet. With only frontend features from HLS, XPNet combines Transfer-Learning with innovative data selection techniques that enable efficient fine-tuning for a new target FPGA. With XPNet, models trained on one FPGA can be quickly adapted to a new target FPGA and used to efficiently predict the power on this new FPGA with high accuracy. Experiments with Polybench, Machsuite and CHStone demonstrate an average error of only 8.40% (10.34% if cross-vendor) when less than 1% of designs are used for the fine-tuning to the new target FPGA. In comparison to best prior model (with full training and 5.74% error), XPNet yields 232x speed up in dataset preparation and training, and 5x speed up in inference on a new FPGA.

***Index Terms*—High Level Synthesis, Machine Learning, FPGA, Power Estimation**

TABLE I
THE FEATURES USED BY PRIOR POWER MODELS VS XPNET

	HLS frontend		HLS backend	
	IR	toggling	binding info	scheduling info
Prior work [2] [3]	✓	✓	✓	✓
XPNet	✓	✓	✗	✗

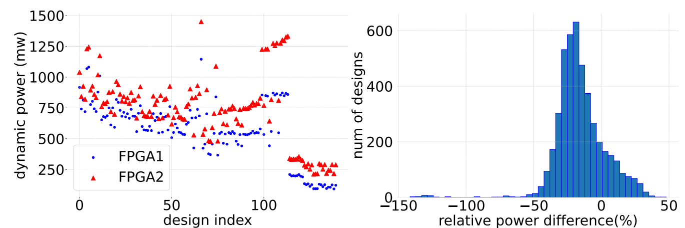


Fig. 1. (Left) Power consumption of multiple designs on 2 different FPGAs. For each design, blue represents one FPGA and red represents another. (Right) Histogram showing the power difference of each design between 2 different FPGAs.

I. Introduction

High Level Synthesis (HLS) [1] is an automated flow to transform an application written in a high-level language, such as C/C++ or Python, to a register-transfer level (RTL) description. Then, another automated process including logic-synthesis, placement and routing converts the RTL into an implementation suitable for field-programmable gate arrays (FPGAs). HLS is composed of two phases: front-end and back-end. The front-end compiles the C/C++ and HLS pragmas into HLS LLVM Intermediate Representation (IR) codes. The back-end will then make FPGA-specific optimizations on the IR code. Resource allocation, operator scheduling and binding happen in this stage.

Although HLS is faster, the logic synthesis, placement and routing phases are much slower, which hinders quick and broad design space exploration. In addition, evaluating power involves writing stimulus and performing simulations, which requires even more time and effort.

To enable accurate and fast estimation of power, prior work employed Machine Learning (ML) techniques to bypass these time-consuming phases [2] [3]. ML models are trained with features extracted from the HLS phase and labels from simulation-based power consumption. Using these ML models, post-implementation power can be inferred right after HLS stage. However, these ML-based power models are inefficient. We will explain the source of this inefficiency with three challenges in the following paragraphs.

First, the dataset preparation is time consuming. Obtaining accurate ML-based power models requires a large number of designs (e.g. thousands). Collecting data from these designs can take days to weeks [4]–[7], depending on the computation resources available, because of the time-consuming steps from RTL-description to FPGA implementation and finally simulation-based power evaluation. This long turn-around time becomes a significant obstacle to building an ML model for an FPGA.

Second, current ML-based power models need to be trained from scratch for a new FPGA target. We cannot use a pretrained FPGA model on the other FPGA directly because these models are trained on a dataset of HLS features (from a specific set of workloads) and the power consumption (for a specific FPGA device). Therefore, even if the set of workloads remains the same, the power model cannot be directly reused for the power estimation of a different FPGA hardware because the trained model does not comprehend the

Authorized licensed use limited to: University of Texas at Austin. Downloaded on March 05,2026 at 21:28:52 UTC from IEEE Xplore. Restrictions apply.
© 2025 IEEE. All rights reserved, including rights for text and data mining and training of artificial intelligence and similar technologies. Personal use is permitted,
but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.

This article has been accepted for publication in IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems. This is the author's version which has not been fully edit
content may change prior to final publication. Citation information: DOI 10.1109/TCAD.2025.3646555

new FPGA's architecture, thus, limiting their reuse. The power value will then certainly be different between the two FPGAs as shown in the left of Figure 1. The relative power difference distribution in the right Figure 1 shows that simple scaling or calibration is not sufficient for the power model on one FPGA to function correctly on the other. This problem is exacerbated when FPGAs from multiple vendors are considered, because they can have significantly different architectures. We need a better way to solve this problem.

Third, the prior methods used to build ML models are vendor-specific. The ML-based power model is constructed with the features extracted after HLS backend as summarized in Table I. The features, as well as the way to extract those features, however, are quite specific to AMD/Xilinx vendors. The process cannot be applied to other tools such as Intel HLS, since the required scheduling/binding info cannot be accessed or they are in different format. Therefore, researchers have to spend a significant amount of time to do feature engineering for a new vendor. Moreover, the HLS backend features require extra time to extract which leads to extra time in inference. Our approach, however, employs only the HLS frontend features, which are obtained quickly and are vendor independent.

Furthermore, it would be beneficial to estimate the power consumption of an HLS design on an FPGA that differs from the one available to you. In other words, when using ML, it would be very useful if we can find a way to reuse and adapt a pretrained FPGA-specific power model to predict the power consumption on a different FPGA using a very limited dataset size. Such a methodology could have many use cases:

- Consider that a company is launching a new FPGA. They have a model for power prediction for their older architecture. They can incrementally train this model using a few engineering designs of the new FPGA, and share this model with customers. The customers can use this to plan their systems and software with the new FPGA, and reduce time-to-market for their products.
- Having limited access to physical chips is a common occurrence, whether in academia or in industry. This can be because of chip shortages from supply-chain issues, or just because the chip is in heavy demand and each person/team gets to use it only for a limited amount of time. In this case, a model that can be trained quickly for a new device is very useful for system design.
- Consider an academic researcher proposing a new accelerator but they have power results for an FPGA they own. They want to compare their results to another paper, but the paper used a different FPGA to which they do not have access. They have the power prediction model for their own FPGA. They can incrementally train it for the FPGA used by the other paper using limited designs and predict the power consumption of their accelerator on the other FPGA, and then compare results.

Cross-FPGA power prediction is thus a very important and pertinent usage scenario.

Addressing the above challenges, and with the objective to enable fast adaption from a base learner (power model on one FPGA) to a target learner (power model on the other FPGA, even from another vendor) with very limited dataset size, we propose a novel cross-FPGA power modeling methodology XPNet. By utilizing only HLS frontend features, we are able to build an GNN-based power model that can be easily transferred between FPGAs, including FPGAs from different vendors. XPNet also introduces an innovative power data selection technique that enables efficient and fast ML-based power model adaption. Our contributions in this work can be summarized as follows:

- We demonstrate XPNet, the first methodology which is capable of constructing an ML model that accurately estimates dynamic power **across** FPGAs. By employing only the features from HLS frontend as indicated in Table I, XPNet constructs models that can be transferred between FPGAs, even those from different vendors.
- We propose a GNN-based power model which utilized only HLS front-end features to predict power and the model can be adapted to FPGAs from different vendors.
- We propose a design selection method to select a small set of designs for fine-tuning with the help of a bridge FPGA. With these selected power-informative designs, the model after transfer-learning is able to provide a better result than K-means and random selection methods.
- With XPNet, we are able to construct ML-based models that produce 8.40% average error with pretrained model on one device and 20 designs on the other target device within AMD/Xilinx vendor family (Xilinx-to-Xilinx). XPNet further shows a 10.34% average error predicting using a pretrained model on one AMD/Xilinx device and 20 designs on a target Intel device (Xilinx-to-Intel).
- We evaluate time-to-result, that is, the total time involved in generating the dataset, training the model, and performing predictions. We observe a speedup of 232x compared to a non-fine-tuning based approach and a speedup of 5x in power inference compared to prior work.

The rest of this paper is organized as follows: Section II provides the background and related work. We define the problem in Section III. A high level view of XPNet is provided in Section IV. It includes the model construction method as well as the sample selection mechanism. In Section V, we present how we prepare the datasets and conduct the experiments and we discuss the experimental results. We conclude with a summary of this work in Section VI.

II. Related Work

Machine Learning for HLS Tasks. Machine Learning (ML) algorithms have gained popularity in the Electronic Design Automation (EDA) domain due to their extremely high efficiency, high quality [8], and owing to their great potential to solve NP-complete problems which are common in EDA domain. While traditional analytical solutions, on the other hand, lead to huge time and resource consumption. ML models have shown remarkable success in various design phases of the EDA flow, such as High-Level Synthesis (HLS), [2], [3], [9]–[16], logic synthesis [17], [18], and placement

Authorized licensed use limited to: University of Texas at Austin. Downloaded on March 05,2026 at 21:28:52 UTC from IEEE Xplore. Restrictions apply.
© 2025 IEEE. All rights reserved, including rights for text and data mining and training of artificial intelligence and similar technologies. Personal use is permitted,
but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.

This article has been accepted for publication in IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems. This is the author's version which has not been fully edit
content may change prior to final publication. Citation information: DOI 10.1109/TCAD.2025.3646555

and routing in physical design [19]–[23]. As [8] points out there are four major tasks specifically for HLS: (1) Result prediction including timing, resource usage, power, maximum frequency, throughput, area, latency and operation delay [12], [16], [24]; (2) Cross-platform performance prediction such as performance prediction for new FPGA platforms and performance prediction for new applications through the execution on CPUs [14], [25]; (3)Active Learning where DSE (Design Space Exploration) for HLS is performed and ML models are used as surrogates for actual synthesis when evaluating a design [11], [26]; (4) Improving optimization algorithms where ML models are used to substitute traditional algorithms for hyper-parameter or configuration selection [27], [28].

Transfer learning has emerged as a promising technique in high-level synthesis (HLS) to address the challenges of limited labeled data and the high cost of data collection for related HLS tasks. In HLS-based design flows, performance metrics such as power, area, and timing are sensitive to both the design characteristics and the synthesis toolchain. Training machine learning models from scratch for each new design or toolchain is often time-consuming and resource-intensive. Transfer learning alleviates this burden by leveraging knowledge from a source domain—such as previously characterized designs—to improve learning efficiency and enhance model generalization in a different but related target domain. Recent studies have demonstrated the effectiveness of transfer learning in accelerating design space exploration (DSE) across different HLS designs [29] and toolchains [30].

Machine Learning for Power Estimation. Traditional accurate power analysis is usually inefficient due to long-running synthesis and simulation. Power is computed from the switching activities of individual signal nets and the capacitive load they drive. The approach is very accurate and serves as the sign-off standard; however, it comes with a very long turn-around time in simulation and computational cost. To address the problem, ML techniques have been widely employed in every design phase to perform power estimation including architecture-level power prediction [31]–[33], RTL stage power modeling [34]–[42] and HLS stage power estimation [2], [3], [14]–[16], [29], [43]. Compared to architecture-level power estimation, HLS designs provide a closer look at the hardware designs, and thus more accurate power estimation can be produced. Although RTL-based power evaluation is more accurate, HLS-level power estimation can save significant time without losing much fidelity of results. HL-Pow [2] adopts Convolutional-Neural-Networks (CNNs) to infer the measured power onboard. They generate the switching activity on the C-level operators and further link them to RTL operators with HLS report mapping information. The switching histogram is built for each operator and fed into their CNNs to infer the power. PowerGear [3], on the other hand, uses graph-neural networks (GNNs) to perform the estimation. They extract the switching activities in a similar way as HL-Pow and recover a graph with operators and the switching characteristics. The graph samples are then used in a GNN model to infer the dynamic power. Unlike the previous two works, HLSPredict [14] uses Random Forest (RF) and Artificial Neural Network (ANN) to predict the power of HLS design with performance counters on a desktop CPU as features.

Through careful examination of prior work and preliminary experiments, we observe that generating a comprehensive dataset for a single FPGA can be extremely time-consuming—often taking weeks or even months. This makes it impractical to curate a new dataset each time power prediction is needed for a different FPGA. Therefore, there is a clear need for a method to build machine learning-based power models that can be reused efficiently across multiple FPGA platforms. XPNet is developed with this goal in mind. While it draws inspiration from prior works such as PowerGear [3], HARP [10], and IronMan [13] in how they construct feature graphs for HLS designs, XPNet is specifically designed to focus on power prediction across a wide range of FPGAs. The following sections detail the design and functionality of XPNet.

III. Problem Formulation

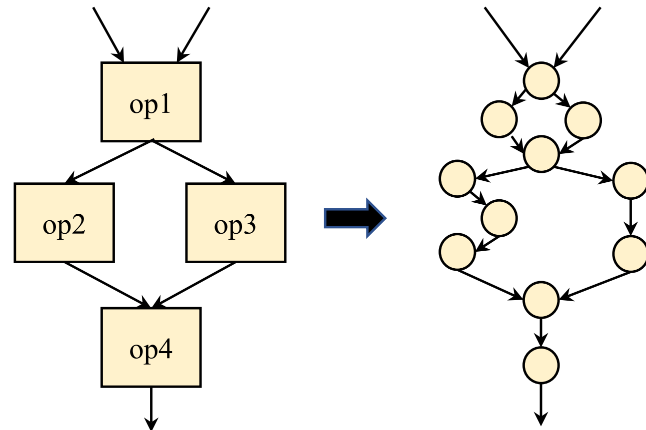


Fig. 2. Graph generated after HLS (left) and graph generated after implementation (right)

In the flow of mapping C applications to an FPGA, two main graphs are generated by the High Level Synthesis (HLS) and implementation (i.e. placement and routing) tools. As illustrated in Figure 2, the coarse-grained high-level operator graph is generated after HLS, and each node represents one C-level operator and each edge represents the data flow between operators. The fine-grained or circuit-level graph is generated after implementation, and each node represents one logic gate and each edge represents the wire between two logic gates. The ground truth power is calculated based on the fine-grained graph with the Equation 1:

$$P_{dyn} = \sum_{i \in I} \alpha_i C_i V^2 f \tag{1}$$

In the equation, α_i is the signal switching activity, C_i is the interconnect capacitance, V is the supply voltage, f is the operating frequency and i is an interconnect of the whole set I.

Authorized licensed use limited to: University of Texas at Austin. Downloaded on March 05,2026 at 21:28:52 UTC from IEEE Xplore. Restrictions apply.
© 2025 IEEE. All rights reserved, including rights for text and data mining and training of artificial intelligence and similar technologies. Personal use is permitted,
but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.

This article has been accepted for publication in IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems. This is the author's version which has not been fully edi
content may change prior to final publication. Citation information: DOI 10.1109/TCAD.2025.3646555

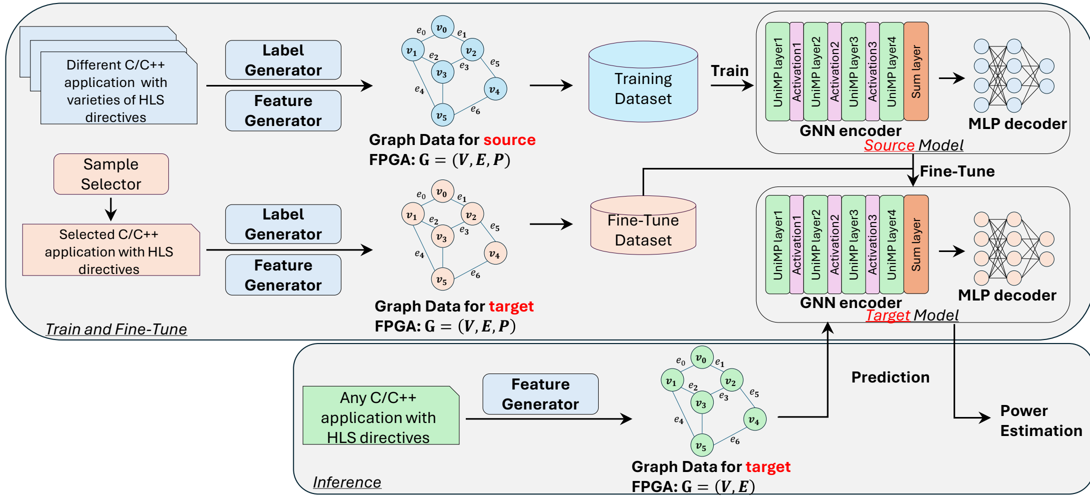


Fig. 3. Overview of XPNet. Train and Fine-Tune (top). Inference (bottom)

For a specific FPGA, the supply voltage V is fixed, and C for each i is decided by the FPGA resources type and placement and routing algorithms.

While different mapping algorithms and optimization strategies can lead to different circuit-level graphs generated from the same high-level graph, it is assumed that the optimization and mapping algorithm are fixed in the problem and dataset generation, therefore, each HLS-level graph can only generate one unique circuit-level graph.

Formally, let m be the total number of HLS-level graphs in the training set. Define $X = (x_1, \cdots, x_m)$ and $Y = (y_1, \cdots, y_m)$, where $x_j \in (\mathbb{R}^d, (V_j, E_j, R_j))$ denotes the global metadata and graph data obtained and preprocessed from HLS-level graph, V_j denotes set of nodes, E_j denotes set of edges and R_j denotes set of relation types on all edges, $\{\forall e \in E_j, \forall v \in V_j, \forall r \in R_j, e, v \in \mathbb{R}^d, r \in \mathbb{R}\}$, and $y_j \in \mathbb{R}$ denotes the power obtained from circuit-level simulation. The goal is to find a hypothesis $g : \mathbb{R}^d, (V, E, R) \rightarrow \mathbb{R}$ so that the mean absolute percentage error $err = \frac{1}{m}\sum_{i=1}^{m} |\frac{g(x_i)-y_i}{y_i}|$ is minimized. It can be intuitive to see from Equation 1 that the power is a linear combination of signal activities α_i and interconnect capacitance C_i and the task can be defined as a regression problem. However, features extracted from HLS-level graph are not necessarily linear mapping to α_i and C_i. Therefore, the solver should not be restricted to be linear.

It is worth mentioning that the above (X, Y) and the outcome hypothesis g are specific to one FPGA. Even though the HLS-graph for the new FPGA is not subject to change, the ground-truth power can still be vastly different due to the great difference on voltage supply V, capacitance C_i and achieved frequency f on a different FPGA. Therefore, given (X', Y'), the hypothesis g cannot achieve the same accuracy as it does on (X, Y). Intuitively, the task to solve on (X, Y) should be quite similar to the task on (X', Y'). The objective of this work is to find a hypothesis g' with as few as possible data samples drawn from (X', Y') given pretrained model g on (X, Y) without much accuracy loss.

IV. The XPNet Framework

Figure 3 depicts a high-level overview of XPNet which is a machine learning framework for cross-FPGA power prediction. In *Train and Fine-tune*, shown in the top part of the figure, we form the dataset with the help of **Label Generator** and **Feature Generator** so that each sample can be directly digested by a regression ML model. The source FPGA model is trained with this dataset. With the **Design Selector**, we select designs that are the most power-informative (described in section IV-A) to form our fine-tune dataset. We then fine-tune the source model to generate our target model. In *Inference*, shown in the bottom part of the figure, we use our Feature Generator to generate features and these features are fed into the target FPGA model produced during *Train and Fine-tune* to predict the power for target FPGA.

A. Train and Fine-Tune

Train and Fine-Tune contains three important components: Label Generator, Feature Generator and Sample Selector.

Label Generator: As shown in Figure 4, the Label Generator is essentially composed of HLS, place and route, simulation and power analyzer from different vendors. While the ground-truth average dynamic power can be collected from either vector-based power analysis or vector-less power analysis, we have to include correct toggling features which will be discussed in **Feature Generator**. In order to generate sufficient designs, we use the Polybench and Machsuite subset from HLSDataset [4] and the provided scripts to extend the dataset to cover more FPGAs. Specifically, we apply pragmas to each kernel in the benchmark suites, - *array_partition loop_unrolling* and *loop_pipeline* - from the AMD/Xilinx tool suite to generate varieties of HLS designs. We use equivalent or similar pragmas from Intel HLS compiler - *hls_numbanks*, *unroll* and *disable_pipeline* - for generating designs for an Intel

Authorized licensed use limited to: University of Texas at Austin. Downloaded on March 05,2026 at 21:28:52 UTC from IEEE Xplore. Restrictions apply.
© 2025 IEEE. All rights reserved, including rights for text and data mining and training of artificial intelligence and similar technologies. Personal use is permitted,
but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.

This article has been accepted for publication in IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems. This is the author's version which has not been fully edi
content may change prior to final publication. Citation information: DOI 10.1109/TCAD.2025.3646555

FPGA. With this method, we generated a variety of designs for each FPGA.

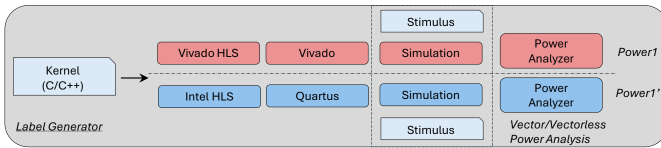


Fig. 4. Detailed view of the Label Generator

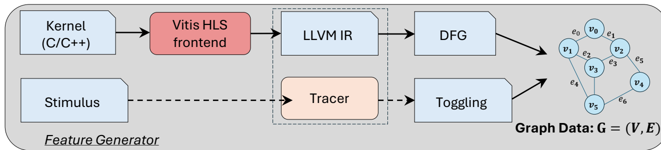


Fig. 5. Detailed view of the Feature Generator

Feature Generator: The detailed overview of Feature Generator is illustrated in Figure 5. We first use the open-source Vitis HLS frontend to compile the kernel code with pragmas into LLVM IR. Then we recover the dataflow graph (DFG) with the LLVM IR. If a stimulus is provided, we parse and annotate the IR code with a trace function to print out the toggling of each IR operator during the execution. In this way, the toggling behavior on each port of the IR operator is recorded. Since the different operators can consume different power even with the same toggling behavior, we need to categorize the collected toggling behavior accordingly. Up to 21 types of operators including logic operators, arithmetic operators, arbitration operators as well as memory operators from HLS are selected. They are summarized as follows:

- Arith: add, sub, mul, div, sqrt, fadd, fsub, fmul, fdiv, fsqrt
- Logic: and, or, xor, icmp, fcmp
- Mem: store, load, read, write
- Arbit: mux, select

The specific method to record and trace IR operator toggling activities has been sufficiently discussed by HL-pow [2] and Dongwook et al. [43]. We further create the graph data using the DFG and toggling activities. The power-aware design representation is composed of two parts. The first is the design itself including the number and types of operators and the path to link these operators. The second component is the behavior of these operators which is normally represented as switching activities. The design generated by HLS tools, however, is determined by two input components. The first one is a high-level program description, expressed in C/C++, which define the semantics and functionality. The second is a set of pragmas that include parallelizing, pipelining and array partitioning. Therefore, in order to correctly represent the two factors, LLVM IR codes generated by HLS tools are chosen. Unlike previous work, where toggling behaviors are collected and mapped to each RTL operator using HLS backend information, our approach maps toggling activities directly to each IR operator. This avoids the time-consuming process of backend-based mapping and improves compatibility across different HLS toolchains. Similarly in [3], [43], we recover the data flow graphs (DFGs) with the LLVM IR codes generated from HLS tools and switching activities obtained from IR-level simulation. Specifically, given a graph $G = (V, E)$, where V and E represent the node and edge set, respectively. Every node in the graph $\forall v \in V$ represents an IR operator with attributes that include opcode type, bit width, input, and output switching activities. Every edge in the graph $\forall e_{i,j} \in E$ where $e_{i,j}$ is the edge with i as the source and j as the sink. The edge $e_{i,j}$ contains switching activities $SA_{i,j}$ and the activation ratio $AR_{i,j}$:

$$SA_{i,j} = \frac{\sum HD(v_i(k), v_i(k-1))}{L}, \quad AR_{i,j} = \frac{N}{L} \tag{2}$$

where HD refers to hamming distance, L refers to the latency of the design and N refers to the number of execution cycles that cause the change of the vertices v_i. The HD inside SA accumulates in every cycle when the vertices change. In addition to these two switching features, the edge type is also encoded and added to the edge attributes.

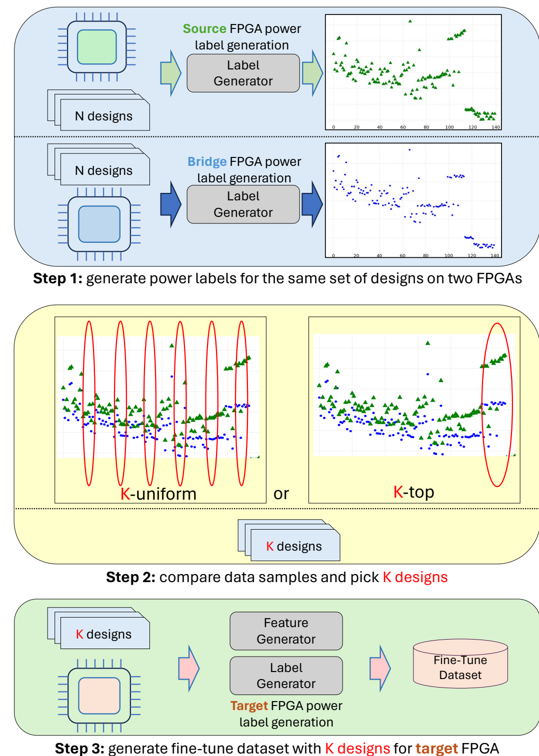


Fig. 6. Detailed view of the Design Selector for fine-tuning dataset

Design Selector: The detailed view of our Design Selector is shown in Figure 6. Different from common data pruning methods which pick samples from the whole dataset, we aim to find the K designs that are power-informative for the target FPGA without generating the whole dataset for the target FPGA. Achieving this goal requires several steps in our Design Selector: *Step 1:* Generate abundant data samples for

Authorized licensed use limited to: University of Texas at Austin. Downloaded on March 05,2026 at 21:28:52 UTC from IEEE Xplore. Restrictions apply.
© 2025 IEEE. All rights reserved, including rights for text and data mining and training of artificial intelligence and similar technologies. Personal use is permitted,
but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.

This article has been accepted for publication in IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems. This is the author's version which has not been fully edit
content may change prior to final publication. Citation information: DOI 10.1109/TCAD.2025.3646555

source FPGA and an extra FPGA (bridge FPGA) with the same designs (same C/C++ codes and HLS pragmas). *Step 2:* Compare the data samples from source FPGA and bridge FPGA to identify the K representative designs. *Step 3:* Use those designs to generate fine-tune data samples for target FPGA. Within the *Step 2*, we calculate the relative power difference of each design on the source FPGA and the bridge FPGA as: $d = \frac{|P_{source} - P_{bridge}|}{P_{source}}$. With the power difference metrics, we use two different methods to identify K designs. ***K-top*** method sorts the designs according to absolute relative power difference $|d|$ and picks K designs with largest power difference value. The K designs are then used to generate data samples from target FPGA. On the other hand, the ***K-uniform*** method sorts the designs according to relative power difference d and picks K designs evenly distributed within the range of the power difference, e.g., if the power difference of the designs range from -10% to 10% and we want to pick 4 designs, then we will randomly pick one design from each chunk of $[-10\%, -5\%]$, $[-5\%, 0\%]$, $[0\%, 5\%]$, $[5\%, 10\%]$. The final model generated by XPNet can then be used directly to predict power for a vast array of designs on the target FPGA.

B. Inference

The process in *Inference* is straightforward; we use the Feature Generator to generate features for the design. We can choose to include the stimulus for the kernel so that the Feature Generator can take the toggling info from running the kernel. Alternatively, users can enter the preferred constant toggling rate into the Feature Generator. These features are fed into the target FPGA model which is produced in the *Train and Fine-tune* step to generate the average power prediction for the design. XPNet only requires kernel program with pragmas and stimulus running on that kernel to predict the average power.

C. Model

The model is composed of a GNN encoder and an MLP decoder for regression task. The GNN encoder can help to better extract the inherent information of the design since edge attributes, including switching features, are essential to perform power prediction. However, both GCN [44] and GAT [45] overlook the edge embeddings. Although PowerGear [3] proposes an edge-expressive GNN, the convolution is performed on each node with neighbor edges where the neighbor nodes and further edges are not fully utilized. Moreover, the number of learnable edge weights is restricted by the number of edge relation types that cannot represent varieties of capacitance in circuits. UniMP [46], inspired by Transformer [47] uses a different aggregation mechanism on each edge. It builds attention coefficients $\alpha_{i,j}$ with both edge and node attributes in every UniMP layer:

$$\alpha_{i,j}^{(l)} = softmax\left(\frac{(W_1^{(l)} h_i^{(l)})^T (W_2^{(l)} h_j^{(l)} + W_3 e_{i,j})}{\sqrt{D}}\right) \quad (3)$$

where l refers to the layer, $e_{i,j}$ represents the edge pointing from vertices v_i to v_j, h_i refers to the node embedding at vertices v_i, D is the hidden size of each head. In the end of the layer, each node embedding is updated with message aggregation from the distant j to the source i:

$$h_i^{(l+1)} = \sum_{j \in N(i)} \alpha_{i,j}^{(l)} \left(W_4 v_j^{(l)} + e_{i,j} \right) \quad (4)$$

To generate one vector representation h_G for the entire graph, we aggregate all the node embeddings for every UniMP layer in the last sum layer as follows:

$$h_G = \sum_{l \in L} \sum_{v \in V} h_v^{(l)} \quad (5)$$

where L is the set of indexes of GNN layers and V is the set of vertices in the graph. The aggregation of all nodes across layers can enhance the generalization ability of the model. Our GNN model is made up of 4 UniMP layers, 3 ReLU activation layers, and 1 sum layer. During fine-tuning, we freeze the GNN encoder and only update the MLP decoder to improve the efficiency.

TABLE II
OVERVIEW OF CHARACTERISTICS OF FPGAS USED IN EXPERIMENTS

AMD/Xilinx					
Name	**Device**	**Tech node**	**#LUTs**	**#BRAMs**	**#DSPs**
FPGA1	xczu9eg	16nm	274,080	912	2,520
FPGA2	xc7v585t	28nm	364,200	795	1,260
FPGA3	xcvu440	20nm	2,532,960	2,520	2,880
FPGA4	xc7k480T	28nm	298,600	955	1,920
FPGA5	ku115	20nm	663,360	2,160	5,520
Intel					
Name	**Device**	**Tech node**	**#ALMs**	**#M20K**	**#DSPs**
FPGA6	S10TX	14nm	449,280	5,461	2,592
FPGA7	A10GX	20nm	339,620	2,423	1,518

V. EVALUATION

A. Experimental Setup

Devices: As Table II shows, we pick five devices from three families of Xilinx: 7 series, Ultrascale and Ultrascale+. Two devices from Intel are selected: Stratix 10 TX and Arria 10 GX. As can be seen in Table II, these devices are quite different from each other and we believe that they should be sufficient to demonstrate the effectiveness of XPNet.

Dataset: We use the Feature Generator and Label Generator to form our training dataset. AMD/Xilinx Vivado HLS 2018.3, Vivado 2018.3, quartus 2019 are used to synthesize and simulate the design to collect the design features and ground-truth power. Similar to the method in PowerGear [3], we generate more than 5000 designs for each FPGA using multiple kernels from different benchmark suites including Polybench [48], Machsuite [49] and CHStone [50]. Details are summarized in Table III. We further extend the method to include designs for Intel FPGAs. We split `atax`, `spmv_crs`, `stencil3d`, `stencil2d`, `gsm` and `sha` out from our dataset to form the testing set. Models are tested with the whole testing set unless specified in the section. The rest of the dataset is used for training. The post-implementation resource utilization and latency for the Xilinx designs is illustrated in Fig. 7 and Intel's

Authorized licensed use limited to: University of Texas at Austin. Downloaded on March 05,2026 at 21:28:52 UTC from IEEE Xplore. Restrictions apply.
© 2025 IEEE. All rights reserved, including rights for text and data mining and training of artificial intelligence and similar technologies. Personal use is permitted,
but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.

This article has been accepted for publication in IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems. This is the author's version which has not been fully edi
content may change prior to final publication. Citation information: DOI 10.1109/TCAD.2025.3646555

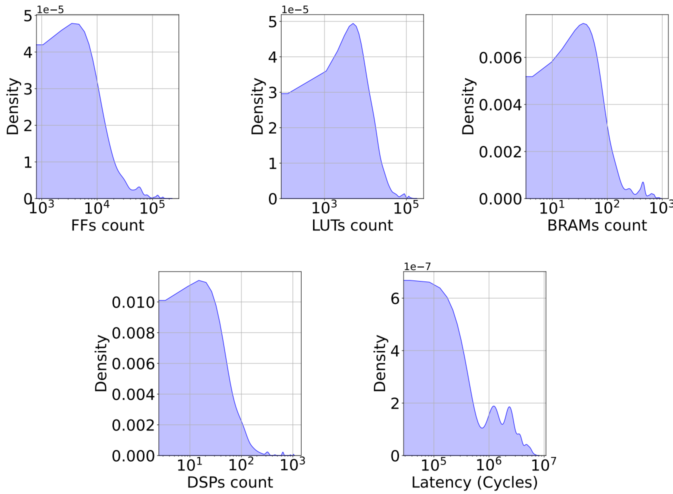


Fig. 7. Resource utilization and latency for the *Xilinx* designs

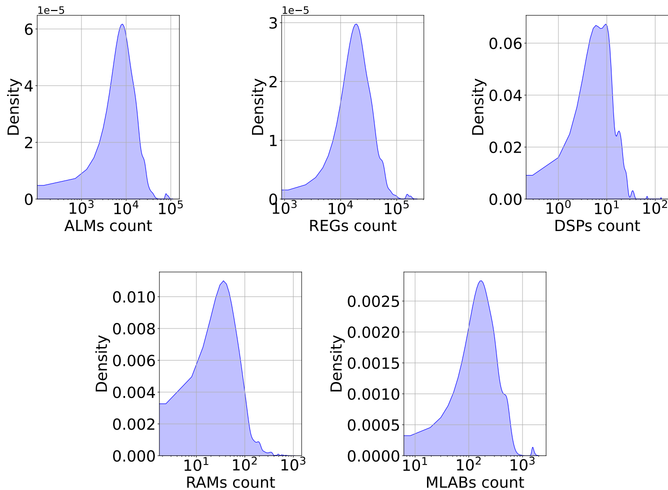


Fig. 8. Resource utilization for the *Intel* designs

is shown in Fig. 8. These designs are used to generate the complete dataset for our experiments.

While the testing set is kept alone, we use 10-fold cross-validation setting across the training dataset during training. All the results are obtained with the evaluation using the test dataset from the target FPGA which we split out in advance. We evaluate the model performance in terms of the mean absolute percentage error (MAPE) unless indicated otherwise in the subsections. The default Design Selector is K-top unless otherwise specified. We used Mean Absolute Error Percentage (MAPE) (μ) and standard deviation (σ) to evaluate the model:

$$e(i) = \left| \frac{y_i - \hat{y}_i}{y_i} \right| \qquad \mu = \frac{1}{n} \sum_{i=1}^{n} e(i) \times 100\% \qquad \sigma = \sqrt{\frac{\sum_i^n (e(i) - \mu)^2}{n}} \tag{6}$$

where n refers to the number of test designs, y_i is the actual power and $\hat{y}_i$ is the predicted power of i_{th} design, $\bar{y}$ is the mean of the actual power.

TABLE III
BENCHMARK KERNELS USED FOR THE DATASET

Benchmark suite	Kernel application	# Designs
Polybench	`atax`, bicg, gemm, gesummv, k2mm, k3mm, mvt, syrk, syr2k	4779
Machsuite	`spmv_crs`, `stencil3d`, `stencil2d`	376
CHStone	`gsm`, `sha`	236

B. How effective is Transfer-Learning in XPNet?

In this section, we showcase the effectiveness of XPNet by comparing it to ordinary (from scratch) training. As shown in Table IV, we conduct two comparison experiments. All models in this section are tested with the designs from `atax`. In the first row, Scratch(3) refers to training the model from scratch with the FPGA3 dataset and testing on FPGA3 dataset, and XPNet(1-3) means training the source model with FPGA1 dataset and fine-tune the source model for FPGA3. FPGA2 is used as the bridge FPGA in these experiments. For various numbers of designs, scratch training uses them for training while XPNet uses them to fine-tune the pre-trained model. Each row contains the model error for the two different methods.

The error remains similar for XPNet over scratch training when the dataset size is large, because a sufficient quantity of designs from the target FPGA can provide good generalization. However, improvements become more obvious when the dataset size decreases since the limited number of designs from the target FPGA fail to fit the power model, when training from scratch. With XPNet, prior knowledge from the source FPGA in the model helps the model to stay accurate even when using only a small number of designs from the target FPGA.

TABLE IV
XPNET VS. TRAIN FROM SCRATCH, MODEL ERROR MAPE(%)

# Designs	Scratch(3)	XPNet(1-3)	Scratch(4)	XPNet(1-4)
20	101.23	8.75	110.63	8.54
50	98.13	8.31	101.32	8.42
100	79.31	8.01	78.01	7.98
200	58.39	7.85	63.32	7.56
300	38.76	7.67	39.19	7.01
400	25.09	7.34	24.43	7.21
500	16.32	7.01	15.13	6.87
1000	10.14	6.76	10.34	6.21
2000	7.86	6.34	8.19	6.13
4000	6.13	6.02	6.45	6.10

C. How effective is XPNet in predicting across devices?

There are two specific questions regarding the generalization of XPNet: *How is the performance of the transferred model impacted when using different source and target FPGAs? How does XPNet perform when source FPGA and target FPGA are from different vendors?* In order to answer these questions, we conduct a set of experiments with different combinations of FPGAs as source, bridge and target. The results of these experiments are shown in Table V and Table VI where s1b2

Authorized licensed use limited to: University of Texas at Austin. Downloaded on March 05,2026 at 21:28:52 UTC from IEEE Xplore. Restrictions apply.
© 2025 IEEE. All rights reserved, including rights for text and data mining and training of artificial intelligence and similar technologies. Personal use is permitted,
but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.

This article has been accepted for publication in IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems. This is the author's version which has not been fully edit
content may change prior to final publication. Citation information: DOI 10.1109/TCAD.2025.3646555

TABLE V
MODELS ON AMD/XILINX DEVICES AS SOURCE MODEL AND XILINX OR INTEL AS TARGET (GNN)

Xilinx-to-Xilinx Transfer model error MAPE(%) and standard deviation across test designs (s=source, b=bridge, t=target)								
Target	**s1b2**	**s1b3**	**s1b4**	**s1b5**	**s2b1**	**s2b3**	**s2b4**	**s2b5**
t1	6.49 (0.038)	6.49 (0.038)	6.49 (0.038)	6.49 (0.038)	NA	8.23 (0.043)	8.54 (0.044)	8.13 (0.042)
t2	NA	8.54 (0.045)	8.87 (0.047)	8.91 (0.044)	6.40 (0.036)	6.40 (0.036)	6.40 (0.036)	6.40 (0.036)
t3	8.75 (0.046)	NA	8.53 (0.042)	8.56 (0.043)	8.35 (0.041)	NA	8.13 (0.046)	8.43 (0.043)
t4	8.54 (0.041)	8.67 (0.047)	NA	8.54 (0.043)	8.24 (0.042)	8.66 (0.042)	NA	8.74 (0.044)
t5	8.23 (0.042)	8.23 (0.043)	8.32 (0.044)	NA	8.02 (0.045)	8.12 (0.043)	8.07 (0.040)	NA
Target	**s3b1**	**s3b2**	**s3b4**	**s3b5**	**s4b1**	**s4b2**	**s4b3**	**s4b5**
t1	NA	8.71 (0.041)	8.14 (0.040)	8.32 (0.042)	NA	8.22 (0.042)	8.33 (0.041)	8.32 (0.042)
t2	8.72 (0.044)	NA	8.13 (0.044)	8.23 (0.046)	8.25 (0.040)	NA	8.65 (0.039)	8.54 (0.040)
t3	6.13 (0.033)	6.13 (0.033)	6.13 (0.033)	6.13(0.033)	8.34 (0.040)	8.24 (0.043)	NA	8.43 (0.041)
t4	8.32 (0.046)	8.34 (0.047)	NA	8.84 (0.047)	6.45 (0.033)	6.45 (0.033)	6.45 (0.033)	6.45 (0.033)
t5	8.45 (0.043)	8.13 (0.041)	8.31 (0.043)	NA	8.05 (0.044)	8.22 (0.042)	8.23 (0.041)	NA
Xilinx-to-Intel Transfer model error MAPE(%) and standard deviation across test designs (s=source, b=bridge, t=target)								
Target	**s1b2**	**s1b3**	**s1b4**	**s1b5**	**s2b1**	**s2b3**	**s2b4**	**s2b5**
t6	9.32 (0.055)	10.12 (0.053)	9.98 (0.054)	10.01 (0.053)	10.01 (0.056)	10.45 (0.057)	10.15 (0.055)	10.23 (0.055)
t7	10.12 (0.056)	10.22 (0.055)	10.17 (0.053)	10.29 (0.053)	10.13 (0.056)	10.23 (0.053)	10.33 (0.057)	10.39 (0.054)
Target	**s3b1**	**s3b2**	**s3b4**	**s3b5**	**s4b1**	**s4b2**	**s4b3**	**s4b5**
t6	10.23 (0.056)	10.13 (0.058)	10.99 (0.057)	10.88 (0.055)	10.33 (0.056)	10.41 (0.057)	10.74 (0.056)	10.41 (0.056)
t7	10.89 (0.057)	10.43 (0.054)	10.87 (0.055)	10.65 (0.056)	10.57 (0.055)	10.65 (0.057)	10.13 (0.056)	10.31 (0.055)

TABLE VI
MODELS ON INTEL DEVICES AS SOURCE MODELS (GNN)

Intel-to-Intel Transfer model error MAPE(%) and standard deviation across test designs (s=source, b=bridge, t=target)										
Target	**s6b1**	**s6b2**	**s6b3**	**s6b4**	**s6b7**	**s7b1**	**s7b2**	**s7b3**	**s7b4**	**s7b6**
t6	7.31(0.029)	7.31(0.029)	7.31(0.029)	7.31(0.029)	7.31(0.029)	8.23(0.038)	8.12(0.039)	7.98(0.038)	8.01(0.037)	NA
t7	7.79(0.037)	8.13(0.038)	8.10(0.037)	8.21(0.038)	NA	8.18(0.030)	8.18(0.030)	8.18(0.030)	8.18(0.030)	8.18(0.030)
Intel-to-Xilinx Transfer model error MAPE(%) and standard deviation across test designs (s=source, b=bridge, t=target)										
Target	**s6b1**	**s6b2**	**s6b3**	**s6b4**	**s6b7**	**s7b1**	**s7b2**	**s7b3**	**s7b4**	**s7b6**
t4	10.45(0.058)	10.78(0.057)	10.98(0.059)	NA	10.87(0.058)	10.31(0.059)	10.14(0.056)	10.69(0.057)	NA	10.13(0.058)
t5	10.14(0.056)	10.65(0.058)	10.42(0.060)	10.88(0.059)	11.12(0.058)	10.09(0.057)	10.63(0.055)	11.03(0.054)	10.86(0.057)	10.79(0.059)

means FPGA1 is used as the source FPGA and FPGA2 is used as the bridge FPGA and t3 means that FPGA3 is taken as the target. Both MAPE and standard deviation across the whole test designs are reported for each run. We also underline the model error number when the target and source are the same; these models do not need a fine-tuning phase. All the other cases require 20-sample fine-tune datasets.

Same-board performance: the model is first tested on the same FPGA and the results are shown with the underlined number in the table. In this case, model is trained and tested on the same FPGA. It shows averagely 6.37% error on 4 Xilinx devices and 7.75% error on 2 Intel devices.

Xilinx-to-Xilinx: when both the source and target FPGA are from Xilinx series, the transferred model error can achieve 8.40% in average. With the same settings of source and bridge FPGA, the model performance is within 1% difference with different target FPGAs. It is also worth mentioning that every FPGA is in a different technology node from each other as indicated in Table II. There are several ”NA”s in the **t5** row. We do not consider **b5t5** as a valid setting since it requires the whole dataset of the target FPGA which is not the purpose of these experiments. *Xilinx-to-Intel:* When the source FPGA is from Xilinx and target FPGA is from Intel, the transferred model error can achieve 10.34%. A performance drop is observed when the target FPGA is from a difference vendor because when the pre-trained model is created with Xilinx FPGAs and the circuit features are captured from the front-end IR codes which are generated by xilinx-hls-llvm. Therefore, these circuit features can correctly reflect the circuit generated by Xilinx tool chains. Intel tool chains, however, may have different strategies/algorithms in high-level-synthesis so that the circuits generated by Intel cannot be the same as Xilinx even though the same HLS options are used in both cases. Hence some drop in model performance is understandable. *Intel-to-Intel:* When both the source FPGA and target FPGA are from Intel, the transferred model error can achieve 8.07% in average, which is quite close to the results gained from Xilinx-to-xilinx experiments. *Intel-to-Xilinx:* When the source

Authorized licensed use limited to: University of Texas at Austin. Downloaded on March 05,2026 at 21:28:52 UTC from IEEE Xplore. Restrictions apply.
© 2025 IEEE. All rights reserved, including rights for text and data mining and training of artificial intelligence and similar technologies. Personal use is permitted,
but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.

This article has been accepted for publication in IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems. This is the author's version which has not been fully edi
content may change prior to final publication. Citation information: DOI 10.1109/TCAD.2025.3646555

TABLE VII
MODEL ERROR MAPE(%) AND STANDARD DEVIATION ACROSS TEST DESIGNS FOR XPNET WITH DIFFERENT DESIGN SELECTORS

	XPNet (1-3)		
# Designs	**K-top**	**K-uniform**	**K-means**
20	8.75 (0.046)	10.13 (0.052)	12.02 (0.056)
50	8.31 (0.041)	9.98 (0.047)	11.12 (0.048)
100	8.01 (0.037)	9.32 (0.041)	10.87 (0.045)
500	7.01 (0.033)	8.35 (0.036)	8.16 (0.039)
1000	6.76 (0.030)	7.13 (0.032)	7.20 (0.031)
2000	6.34 (0.028)	6.89 (0.028)	6.72 (0.027)
4000	6.02 (0.027)	6.02 (0.027)	6.02 (0.027)
	XPNet (1-4)		
# Designs	**K-top**	**K-uniform**	**K-means**
20	8.54 (0.041)	10.88 (0.046)	6.10 (0.055)
50	8.42 (0.037)	10.23 (0.041)	6.10 (0.047)
100	7.98 (0.033)	9.36 (0.036)	6.10 (0.041)
500	6.87 (0.031)	7.56 (0.032)	6.10 (0.035)
1000	6.21 (0.031)	6.93 (0.031)	6.10 (0.032)
2000	6.13 (0.029)	6.23 (0.029)	6.10 (0.029)
4000	6.10 (0.028)	6.10 (0.028)	6.10 (0.028)

FPGA is from Intel and target FPGA is from Xilinx, the transferred model error can achieve 10.59% where a slight performance degradation happens due to the large difference between two vendors.

Bridge FPGA effect: The bridge FPGA is used when we need to select designs that are used to fine tune the model for the target FPGA, the use of bridge FPGA is to identify the designs that make a difference in the power on two different FPGAs so that those power-informative designs can fine-tune the model better. Therefore, which bridge FPGA to pick is also important. When all the three FPGAs are from the same vendor, the performance of XPNet is quite stable. However, when the source and target FPGAs are from different vendors, the results with the bridge and target FPGA from the same vendor are always better than those with bridge and target FPGAs from different vendors (e.g., **s6b1t4** vs. **s6b7t4**). In order to achieve the best performance, it is the best to pick a bridge FPGA which is similar to the target FPGA.

In summary, with different combinations of source and bridge FPGAs, XPNet performance is not largely affected. XPNet can perform better when the source and target FPGA are from the same vendor while it maintains good performance even when they are from different vendors.

D. How much can the Design Selector help?

Existing work have proposed methods to form core-set for efficient transfer-learning [30], [51]. K-means algorithm is applied in [30] for efficient transfer-learning to solve HLS design space exploration tasks. While K-means algorithm can efficiently find a core-set of a large dataset, it requires the whole dataset as input.

In this section, we evaluate the performance of two design selection methods in XPNet and K-means algorithm as baseline. The K-top and K-uniform selection methods are used to construct the fine-tuning dataset, as discussed in the previous section. The K-means algorithm is used to select K representative designs from the source FPGA dataset. These designs, along with their corresponding pragmas and settings, are then used to generate a new set of K designs on the target FPGA, forming the fine-tuning dataset. Three selectors are evaluated with test set from the target FPGA and their results are shown in Table VII. Among the three methods, K-top performs the best and K-means performs the worst. Although the K-means algorithm can select representative designs from the existing dataset, these designs may not be sufficiently informative for power estimation across different FPGAs.

We also further implement a Random selector as another baseline to imitate the case where transfer-learning is applied without any design selection. Four selection methods are employed: K-top, K-uniform, K-means, and Random. Each selection method is repeated 20 times, resulting in 20 fine-tuning datasets per method. They are evaluated with the same test set from the target FPGA and results are illustrated in Fig. 9. While the random selection method cannot guarantee consistent performance of the transferred model, the other three methods yield more stable results.

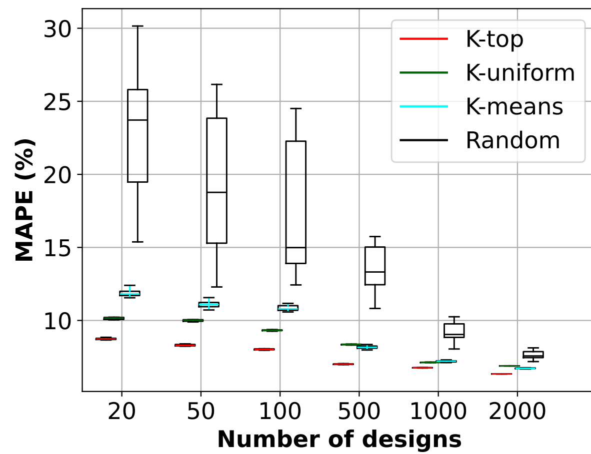


Fig. 9. XPNet (1-3) with the fine-tune dataset created by **K-top, K-uniform, K-means** and **Random** design selection, the fine-turning process with each selection method is repeated 20 times with different seeds

E. Comparison on different ML Models on XPNet

While K-top and K-uniform generate consistent fine-tune dataset every time and the model performance is quite stable, random selector cannot guarantee the same dataset and it is observed that the model performance can vary with different fine-tune dataset. The error increases for all three methods when the number of designs decreases. While the difference between the K-uniform and K-top is slight, the models fine-tuned by random selector cannot guarantee the target model performance since the quality of fine-tune dataset cannot be guaranteed. However, as the fine-tune dataset size increases, random selector tends to provide models with stable perfor-

Authorized licensed use limited to: University of Texas at Austin. Downloaded on March 05,2026 at 21:28:52 UTC from IEEE Xplore. Restrictions apply.
© 2025 IEEE. All rights reserved, including rights for text and data mining and training of artificial intelligence and similar technologies. Personal use is permitted,
but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.

This article has been accepted for publication in IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems. This is the author's version which has not been fully edi
content may change prior to final publication. Citation information: DOI 10.1109/TCAD.2025.3646555

TABLE VIII
MODELS ON AMD/XILINX DEVICES AS SOURCE MODEL AND XILINX OR INTEL AS TARGET (MLP)

Xilinx-to-Xilinx Transfer model error MAPE(%) and standard deviation across test designs (s=source, b=bridge, t=target)								
Target	**s1b2**	**s1b3**	**s1b4**	**s1b5**	**s2b1**	**s2b3**	**s2b4**	**s2b5**
t1	8.58 (0.046)	8.58 (0.046)	8.58 (0.046)	8.58 (0.046)	NA	10.32 (0.054)	10.38 (0.057)	10.21 (0.055)
t2	NA	10.54 (0.056)	10.17 (0.051)	10.65 (0.053)	8.71 (0.047)	8.71 (0.047)	8.71 (0.047)	8.71 (0.047)
t3	10.05 (0.050)	NA	10.78 (0.054)	10.74 (0.056)	10.25 (0.055)	NA	10.14 (0.052)	10.33 (0.053)
t4	10.35 (0.052)	10.43 (0.053)	NA	10.23 (0.048)	10.14 (0.056)	10.22 (0.049)	NA	10.21 (0.054)
t5	10.11 (0.051)	10.13 (0.049)	10.21 (0.053)	NA	10.12 (0.051)	10.23 (0.053)	10.07 (0.054)	NA
Target	**s3b1**	**s3b2**	**s3b4**	**s3b5**	**s4b1**	**s4b2**	**s4b3**	**s4b5**
t1	NA	10.23 (0.050)	10.54 (0.052)	10.76 (0.050)	NA	10.02 (0.049)	10.11 (0.051)	10.12 (0.048)
t2	10.37 (0.056)	NA	10.41 (0.051)	10.57 (0.052)	10.15 (0.054)	NA	10.28 (0.057)	10.13 (0.055)
t3	8.33 (0.044)	8.33 (0.044)	8.33 (0.044)	8.33 (0.044)	10.22 (0.051)	10.18 (0.052)	NA	10.32 (0.055)
t4	10.23 (0.053)	10.45 (0.054)	NA	10.58 (0.056)	8.78 (0.046)	8.78 (0.046)	8.78 (0.046)	8.78 (0.046)
t5	10.65 (0.050)	10.19 (0.051)	10.59 (0.055)	NA	10.08 (0.057)	10.02 (0.051)	10.34 (0.050)	NA
Xilinx-to-Intel Transfer model error MAPE(%) and standard deviation across test designs (s=source, b=bridge, t=target)								
Target	**s1b2**	**s1b3**	**s1b4**	**s1b5**	**s2b1**	**s2b3**	**s2b4**	**s2b5**
t6	11.58 (0.061)	12.23 (0.062)	12.01 (0.059)	12.13 (0.058)	12.15 (0.060)	12.69 (0.061)	12.32 (0.058)	12.43 (0.061)
t7	12.41 (0.061)	12.51 (0.058)	12.87 (0.060)	12.69 (0.059)	12.01 (0.061)	12.32 (0.062)	12.65 (0.062)	12.34 (0.059)
Xilinx-to-Intel Transfer model error MAPE(%) and standard deviation across test designs (s=source, b=bridge, t=target)								
Target	**s3b1**	**s3b2**	**s3b4**	**s3b5**	**s4b1**	**s4b2**	**s4b3**	**s4b5**
t6	12.11 (0.063)	12.21 (0.059)	13.01 (0.062)	12.91 (0.063)	12.38 (0.060)	12.47 (0.062)	12.45 (0.063)	12.54 (0.059)
t7	13.43 (0.064)	12.01 (0.059)	12.78 (0.063)	12.65 (0.064)	12.45 (0.059)	12.54 (0.061)	12.63 (0.059)	12.65 (0.063)

TABLE IX
MODELS ON INTEL DEVICES AS SOURCE MODELS (MLP)

Intel-to-Intel Transfer model error MAPE(%) and standard deviation across test designs (s=source, b=bridge, t=target)										
Target	**s6b1**	**s6b2**	**s6b3**	**s6b4**	**s6b7**	**s7b1**	**s7b2**	**s7b3**	**s7b4**	**s7b6**
t6	9.76(0.048)	9.76(0.048)	9.76(0.048)	9.76(0.048)	9.76(0.048)	10.21(0.052)	10.34(0.056)	10.15(0.051)	10.17(0.055)	NA
t7	9.98(0.054)	10.11(0.055)	9.97(0.051)	10.21(0.053)	NA	10.01(0.049)	10.01(0.049)	10.01(0.049)	10.01(0.049)	10.01(0.049)
Intel-to-Xilinx Transfer model MAPE error (%) and standard deviation across test designs (s=source, b=bridge, t=target)										
Target	**s6b1**	**s6b2**	**s6b3**	**s6b4**	**s6b7**	**s7b1**	**s7b2**	**s7b3**	**s7b4**	**s7b6**
t4	12.43(0.066)	12.69(0.068)	13.02(0.069)	NA	13.31(0.067)	12.13(0.064)	13.14(0.066)	13.01(0.067)	NA	13.32(0.066)
t5	12.41(0.065)	12.78(0.069)	12.98(0.068)	12.77(0.066)	13.28(0.067)	12.14(0.068)	12.87(0.068)	13.22(0.067)	12.88(0.069)	13.11(0.070)

TABLE X
THE TABLE INCLUDING FEATURES BEING USED IN THE MLP MODEL

Type	S-type	Toggling	Clock(ns)
fmul	Arith	[1.505, 0.761, 1.569]	10
fmul	Logic	[3.305, 1.732, 0.259]	10
fmul	Mem	[2.345, 0.921, 0.337]	10
fmul	Arbit	[1.305, 0.513, 3.421]	10
fadd	Arbit	[1.533, 1.569, 1.539]	10
...	...	...	..

mance. The model performance difference by three selectors become more obvious when the number of designs decreases, and the design selector by XPNet always performs better than random selector.

We have also implemented an MLP to test the usability of our methodology and the results are shown in Table VIII and Table IX. The flow to generate the model for target FPGA is illustrated in Fig. 10. In order to make the data compatible with the MLP model, we tabularize the activity/toggling info according to each operator's type to create a feature table. **Toggling** is recorded using the same method as discussed in [3], [43]. A snippet of such a feature table can be seen

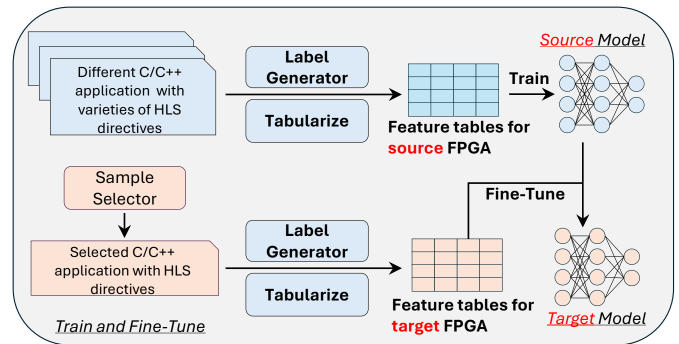


Fig. 10. XPNet implemented with the MLP

Authorized licensed use limited to: University of Texas at Austin. Downloaded on March 05,2026 at 21:28:52 UTC from IEEE Xplore. Restrictions apply.
© 2025 IEEE. All rights reserved, including rights for text and data mining and training of artificial intelligence and similar technologies. Personal use is permitted,
but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.

This article has been accepted for publication in IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems. This is the author's version which has not been fully edi
content may change prior to final publication. Citation information: DOI 10.1109/TCAD.2025.3646555

in Table X. **Type** column refers to the operator type. **S-type** column refers to the operator type that this operator is succeeded by. For example, in the first row, fmul is succeeded by an Arithmetic operator. We further categorize each operator according to its **S-type**, we then aggregate the toggling values (two input and one output) for the operator with the same **Type** and **S-type**. The aggregated values are put in the **Toggling** column, we assume each operator contains two input toggling and one output toggling values. We fill the **Clock** column with the target clock period.

Compared to the GNN implementation, XPNet with MLP has on average 2% degradation. The feature table is generated manually to represent the HLS design. Compared to the GNN encoder, the feature table lacks of important data flow features and it leads to a worse representation of designs. It is also worth mentioning that XPNet with MLP gives the error for Xilinx-to-Xilinx transfer with 10.32% error on average while GNN gives 8.40%. For Xilinx-to-Intel transfer model, MLP can achieve 12.46% error on average vs. GNN 10.34% on average. With the Intel FPGA as source, MLP can achieve 10.14% error and 12.86% error with Intel-to-Intel and Intel-to-Xilinx transfer model respectively, however, GNN can achieve 8.07% and 10.59% accordingly.

F. How does XPNet with different architectures perform on different test designs?

TABLE XI
MAPE (%) AND STANDARD DEVIATIONS OF DIFFERENT MODEL ARCHITECTURES. EACH MODEL IS TRAINED AND TESTED USING S1B2T3 SETTING. LEAVE-ONE-OUT STRATEGY IS APPLIED FOR DATASET SPLIT ON EACH ROW.

Test designs	XPNet encoder layer settings			PowerGear
	GCN	PowerGear	UniMP	
Default	10.67(0.067)	9.14(0.052)	8.75(0.046)	7.65(0.037)
Atax	10.23(0.061)	10.07(0.057)	8.14(0.043)	6.98(0.032)
Bicg	10.91(0.064)	9.78(0.053)	8.53(0.048)	7.04(0.035)
Gemm	11.08(0.062)	9.23(0.050)	8.95(0.051)	7.55(0.039)
Gesummv	10.79(0.064)	10.02(0.059)	8.34(0.045)	8.61(0.044)
K2mm	10.96(0.065)	9.54(0.049)	8.11(0.042)	7.13(0.031)
K3mm	10.43(0.061)	9.88(0.051)	8.32(0.046)	7.42(0.035)
Mvt	10.67(0.068)	10.01(0.053)	7.96(0.041)	6.75(0.029)
Syrk	10.43(0.070)	9.76(0.048)	7.87(0.042)	6.65(0.033)
Syr2k	11.09(0.062)	9.98(0.051)	8.02(0.043)	6.87(0.030)
Average	10.73(0.064)	9.74(0.052)	8.30(0.045)	7.27(0.035)

We further implement XPNet with various graph encoders to demonstrate the effectiveness of the UniMP layer in our proposed XPNet architecture. We implement the GNN encoder with the convolution layer proposed in GCN [44] and PowerGear [3]. The vanilla PowerGear, which used HLS backend features, is also included for comparison. All models are trained and tested with the s1b2t3 setting (source FPGA1, bridge FPGA2, target FPGA3) and top-20 selector. The dataset is split using a leave-one-out strategy, where test designs are set aside in advance for evaluation, and the model is trained on the remaining data.. The MAPE and standard deviation of the model across test designs are reported in the table. XI.

While PowerGear achieves the best performance using backend features, XPNet delivers comparable results with only frontend features. Among the different implementations of XPNet, the version incorporating the UniMP layer performs the best. Compared to GCN, UniMP places greater emphasis on edge features, which are essential for capturing toggling behavior. As a result, UniMP outperforms GCN in this context. Furthermore, unlike the PowerGear convolutional layer, which considers only adjacent nodes and edges, UniMP leverages information from more distant nodes and edges when updating the embedding of each node. Consequently, UniMP also outperforms the use of PowerGear convolutional layers.

G. How does XPNet perform in comparison to other power models?

XPNet has two major advantages: First, it supports cross-FPGA power estimates and the device type is not restricted within one FPGA vendor; Second, it saves a great amount of time for building the model for a new FPGA. We explicitly demonstrate these two advantages in Table XII and Table XIII respectively.

We compare XPNet vs. PowerGear [3], a state-of-art power model, in terms of model performance. However, PowerGear predicts on-board power consumption. So, we adjust the infrastructure slightly so that it can take simulation power as the task, to ensure a fair comparison. With the dataset containing data from Xilinx FPGAs, this PowerGear model error stays at 5.78%. We further integrate PowerGear with our design selector so that it can be adapted to a new FPGA within Xilinx vendor. It is much harder to adjust PowerGear to achieve cross-vendor prediction since the fundamental logic underneath is specific to Xilinx tool chains. To evaluate the stability of our model, each experiment is run 20 times with different random seeds. The average MAPE (%) and standard deviation across these runs are reported. As Table XII shows, PowerGear performs better than XPNet for Xilinx devices because it uses post-HLS features, whereas XPNet uses fewer features and higher-level representation of designs. The use of only higher-level features is also a reason why XPNet can cover many more device types than PowerGear (including those from other vendors).

The efficiency of XPNet to build a model for a new FPGA is analyzed and summarized in Table XIII. For model training, we evaluate the time spent on dataset generation and power model training, and we compare XPNet and PowerGear. The overall time is composed of two parts: dataset preparation time and training time: $T = T_{dataset} + T_{training}$ where $T_{dataset}$ denotes the time to generate data samples for the designs on target FPGA. The time to generate each data sample is composed of HLS time, logic synthesis and implementation time, and simulation time. $T_{dataset}$ is estimated according to the dataset generation running on one Intel Xeon 5218 2.3GHz with 384GB RAM. $T_{training}$ includes training time or fine-tuning time. It is worth mentioning that $T_{dataset}$ can

Authorized licensed use limited to: University of Texas at Austin. Downloaded on March 05,2026 at 21:28:52 UTC from IEEE Xplore. Restrictions apply.
© 2025 IEEE. All rights reserved, including rights for text and data mining and training of artificial intelligence and similar technologies. Personal use is permitted,
but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.

This article has been accepted for publication in IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems. This is the author's version which has not been fully edit
content may change prior to final publication. Citation information: DOI 10.1109/TCAD.2025.3646555

TABLE XII
MODEL ACCURACY COMPARISON: XPNET VS POWERGEAR [3]
AVERAGE MAPE(%) AND STANDARD DEVIATION ACROSS 20 RUNS WITH DIFFERENT SEEDS ARE USED

	Same FPGA		Cross FPGA			
	Xilinx	**Intel**	**Xilinx-to-Xilinx**	**Xilinx-to-Intel**	**Intel-to-Intel**	**Intel-to-Xilinx**
PowerGear	5.78 (0.007)	NA	NA	NA	NA	NA
PowerGear with 20-top Design Selector	5.78 (0.007)	NA	7.26 (0.013)	NA	NA	NA
PowerGear with 50-top Design Selector	5.78 (0.007)	NA	6.31 (0.012)	NA	NA	NA
XPNet with 20-top Design Selector	6.32 (0.011)	7.68 (0.009)	8.56 (0.020)	10.38 (0.023)	8.27 (0.018)	10.71 (0.025)
XPNet with 50-top Design Selector	6.32 (0.011)	7.68 (0.009)	7.45 (0.017)	9.79 (0.019)	7.53 (0.016)	9.56 (0.021)

TABLE XIII
XPNET SPEED COMPARISON TO PRIOR APPROACHES - FOR TRAINING AND INFERENCE

Model for a new FPGA	**Model training**		**Power estimates/inference**	
	Flow	**Runtime(m) / Speedup**	**Flow**	**Speedup**
Traditional analysis model	NA	NA	HLS + Syn, P&R + *Simulation* + Power analyzer	1x
PowerGear	∼5000 designs + Training	82,846 / 1x	HLS + Feature generation	4.8x
XPNet	20 designs + Fine-tuning	356 / 232x	HLS (front-end only) + Feature generation	24x

be reduced with more parallelism and using more powerful machines. Additionally, we do not include the training time of the pretrained model in XPNet. The dataset preparation time on both the source and target FPGAs is also excluded, as this step is performed only once to identify designs on the target FPGAs and is therefore considered as a one-time effort. Due to the existence of prior knowledge in the pretrained model, XPNet can save a significant amount of time than would be required in preparing a large dataset. It turns out XPNet is 232x faster than PowerGear to build a new model. For inference of power estimation on a new design, XPNet, with the invocation of only front-end HLS, is 24x faster than the traditional power analysis method and 5x faster than the state-of-the-art power model.

VI. CONCLUSION

In this work, we propose XPNet, a methodology to build a power model with high level language code that combines transfer learning techniques with novel data sample generation methods on a target FPGA. We showcase that XPNet can build sufficient cross-FPGA power models that can be used to predict power even on FPGAs from different vendors. With the help of transfer learning, K-top sample selection and a bridge FPGA, the GNN based XPNet can achieve 8.40%, 10.32%, 8.07%, 10.59% average error on Xilinx-to-Xilinx, Xilinx-to-Intel, Intel-to-Intel, Intel-to-Xilinx transfer model, respectively. With fewer data samples needed to build a model for a new FPGA, a speedup of 232x is achieved in comparison to the state-of-art power model. Since only front-end HLS features are necessary to build the feature map, a speedup of 5x over the state-of-the-art power model can be observed during inference.

REFERENCES

[1] P. Coussy, D. D. Gajski, M. Meredith, and A. Takach, "An Introduction to High Level Synthesis," in *IEEE Design Test of Computers*, 2009.

[2] Z. Lin, J. Zhao, S. Sinha, and W. Zhang, "HL-Pow: A Learning-Based Power Modeling Framework for High-Level Synthesis," in *Asia and South Pacific Design Automation Conference (ASP-DAC)*, 2020.

[3] Z. Lin, Z. Yuan, J. Zhao, W. Zhang, H. Wang, and Y. Tian, "PowerGear: Early-Stage Power Estimation in FPGA HLS via Heterogeneous Edge-Centric GNNs," in *Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2022.

[4] Z. Wei, A. Arora, R. Li, and L. John, "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)*, 2023.

[5] Y. Bai, A. Sohrabizadeh, Z. Qin, Z. Hu, Y. Sun, and J. Cong, "Towards a comprehensive benchmark for high-level synthesis targeted to FPGAs," in *Thirty-Seventh Conference on Neural Information Processing Systems Datasets and Benchmarks Track*, 2023.

[6] P. Goswami, M. Shahshahani, and D. Bhatia, "MLSBench: A Synthesizable Dataset of HLS Designs to Support ML Based Design Flows," in *Proceedings of the 2020 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, 2020.

[7] L. Ferretti, J. Kwon, G. Ansaloni, G. Di Guglielmo, L. Carloni, and L. Pozzi. DB4HLS: A Database of High-Level Synthesis Design Space Explorations.

[8] G. Huang, J. Hu, Y. He, J. Liu, M. Ma, Z. Shen, J. Wu, Y. Xu, H. Zhang, K. Zhong, X. Ning, Y. Ma, H. Yang, B. Yu, H. Yang, and Y. Wang, "Machine learning for electronic design automation: A survey," *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, 2021.

[9] A. Sohrabizadeh, Y. Bai, Y. Sun, and J. Cong, "Automated accelerator optimization aided by graph neural networks," in *ACM/IEEE Design Automation Conference (DAC)*, 2022.

[10] A. Sohrabizadeh, Y. Bai, Y. Sun, and J. Cong, "Robust GNN-based Representation Learning for HLS," in *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, 2023.

[11] H.-Y. Liu and L. P. Carloni, "On learning-based methods for design-space exploration with High-Level Synthesis," in *ACM/EDAC/IEEE Design Automation Conference (DAC)*, 2013.

[12] E. Ustun, C. Deng, D. Pal, Z. Li, and Z. Zhang, "Accurate operation delay prediction for FPGA HLS using graph neural networks," in *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, 2020.

[13] N. Wu, Y. Xie, and C. Hao, "IronMan-Pro: Multiobjective Design Space Exploration in HLS via Reinforcement Learning and Graph Neural Network-Based Modeling," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, 2023.

[14] K. O'Neal, M. Liu, H. Tang, A. Kalantar, K. DeRenard, and P. Brisk, "HLSPredict: cross platform performance prediction for FPGA high-level synthesis," in *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, 2018.

Authorized licensed use limited to: University of Texas at Austin. Downloaded on March 05,2026 at 21:28:52 UTC from IEEE Xplore. Restrictions apply.
© 2025 IEEE. All rights reserved, including rights for text and data mining and training of artificial intelligence and similar technologies. Personal use is permitted,
but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.

This article has been accepted for publication in IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems. This is the author's version which has not been fully edit
content may change prior to final publication. Citation information: DOI 10.1109/TCAD.2025.3646555

[15] N. Wu, H. Yang, Y. Xie, P. Li, and C. Hao, "High-Level Synthesis Performance Prediction Using GNNs: Benchmarking, Modeling, and Advancing," in *ACM/IEEE Design Automation Conference (DAC)*, 2022.

[16] H. Mohammadi Makrani, F. Farahmand, H. Sayadi, S. Bondi, S. M. Pudukotai Dinakarrao, H. Homayoun, and S. Rafatirad, "Pyramid: Machine Learning Framework to Estimate the Optimal Timing and Resource Usage of a High-Level Synthesis Design," in *International Conference on Field Programmable Logic and Applications (FPL)*, 2019.

[17] C. Yu, H. Xiao, and G. De Micheli, "Developing synthesis flows without human knowledge," in *ACM/IEEE Design Automation Conference (DAC)*, 2018.

[18] 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 *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, 2019.

[19] M. B. Alawieh, W. Li, Y. Lin, L. Singhal, M. A. Iyer, and D. Z. Pan, "High-Definition Routing Congestion Prediction for Large-Scale FPGAs," in *Asia and South Pacific Design Automation Conference (ASP-DAC)*, 2020.

[20] M. Kou, J. Zeng, B. Han, F. Xu, J. Gu, and H. Yao, "GEML: GNN-based efficient mapping method for large loop applications on CGRA," in *ACM/IEEE Design Automation Conference (DAC)*, 2022.

[21] Z. Xie, Y.-H. Huang, G.-Q. Fang, H. Ren, S.-Y. Fang, Y. Chen, and J. Hu, "RouteNet: Routability prediction for Mixed-Size Designs Using Convolutional Neural Network," in *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, 2018.

[22] C.-C. Chang, J. Pan, T. Zhang, Z. Xie, J. Hu, W. Qi, C.-W. Lin, R. Liang, J. Mitra, E. Fallon, and Y. Chen, "Automatic Routability Predictor Development Using Neural Architecture Search," in *IEEE/ACM International Conference On Computer Aided Design (ICCAD)*, 2021.

[23] Y.-H. Huang, Z. Xie, G.-Q. Fang, T.-C. Yu, H. Ren, S.-Y. Fang, Y. Chen, and J. Hu, "Routability-Driven Macro Placement with Embedded CNN-Based Prediction Model," in *Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2019.

[24] S. Dai, Y. Zhou, H. Zhang, E. Ustun, E. F. Young, and Z. Zhang, "Fast and accurate estimation of quality of results in high-level synthesis with machine learning," in *International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, 2018.

[25] H. M. Makrani, H. Sayadi, T. Mohsenin, S. rafatirad, A. Sasan, and H. Homayoun, "XPPE: cross-platform performance estimation of hardware accelerators using machine learning," in *Asia and South Pacific Design Automation Conference (ASP-DAC)*, 2019.

[26] P. Meng, A. Althoff, Q. Gautier, and R. Kastner, "Adaptive Threshold Non-Pareto Elimination: Re-thinking machine learning for system level design space exploration on FPGAs," in *Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2016.

[27] R. G. Kim, J. R. Doppa, and P. P. Pande, "Machine Learning for Design Space Exploration and Optimization of Manycore Systems," in *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, 2018.

[28] Z. Wang and B. C. Schafer, "Machine Leaming to Set Meta-Heuristic Specific Parameters for High-Level Synthesis Design Space Exploration," in *ACM/IEEE Design Automation Conference (DAC)*, 2020.

[29] J. Kwon and L. P. Carloni, "Transfer Learning for Design-Space Exploration with High-Level Synthesis," in *ACM/IEEE Workshop on Machine Learning for CAD (MLCAD)*, 2020.

[30] Z. Ding, A. Sohrabizadeh, W. Li, Z. Qin, Y. Sun, and J. Cong, "Efficient Task Transfer for HLS DSE," in *IEEE/ACM International Conference on Computer Aided Design (ICCAD)*, 2024.

[31] J. Zhai, C. Bai, B. Zhu, Y. Cai, Q. Zhou, and B. Yu, "McPAT-Calib: A Microarchitecture Power Modeling Framework for Modern CPUs," in *IEEE/ACM International Conference On Computer Aided Design (ICCAD)*, 2021.

[32] Y. S. Shao, B. Reagen, G.-Y. Wei, and D. Brooks, "Aladdin: A pre-RTL, power-performance accelerator simulator enabling large design space exploration of customized architectures," in *IEEE/ACM International Symposium on Computer Architecture (ISCA)*, 2014.

[33] Q. Zhang, S. Li, G. Zhou, J. Pan, C.-C. Chang, Y. Chen, and Z. Xie, "PANDA: Architecture-Level Power Evaluation by Unifying Analytical and Machine Learning Solutions," in *IEEE/ACM International Conference on Computer Aided Design (ICCAD)*, 2023.

[34] Z. Xie, X. Xu, M. Walker, J. Knebel, K. Palaniswamy, N. Hebert, J. Hu, H. Yang, Y. Chen, and S. Das, "APOLLO: An Automated Power Modeling Framework for Runtime Power Introspection in High-Volume Commercial Microprocessors," in *IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2021.

[35] Y. Zhou, H. Ren, Y. Zhang, B. Keller, B. Khailany, and Z. Zhang, " PRIMAL: Power Inference using Machine Learning," in *ACM/IEEE Design Automation Conference (DAC)*, 2019.

[36] Z. Xie, S. Li, M. Ma, C.-C. Chang, J. Pan, Y. Chen, and J. Hu, "DEEP: Developing Extremely Efficient Runtime On-Chip Power Meters," in *IEEE/ACM International Conference On Computer Aided Design (ICCAD)*, 2022.

[37] J. Yang, L. Ma, K. Zhao, Y. Cai, and T.-F. Ngai, "Early stage real-time SoC power estimation using RTL instrumentation," in *Asia and South Pacific Design Automation Conference (ASP-DAC)*, 2015.

[38] W. Fang, Y. Lu, S. Liu, Q. Zhang, C. Xu, L. W. Wills, H. Zhang, and Z. Xie, "MasterRTL: A Pre-Synthesis PPA Estimation Framework for Any RTL Design," in *IEEE/ACM International Conference on Computer Aided Design (ICCAD)*, 2023.

[39] P. Sengupta, A. Tyagi, Y. Chen, and J. Hu, "How Good Is Your Verilog RTL Code? A Quick Answer from Machine Learning," in *IEEE/ACM International Conference On Computer Aided Design (ICCAD)*, 2022.

[40] C. Xu, C. Kjellqvist, and L. W. Wills, "SNS's not a synthesizer: a deep-learning-based synthesis predictor," in *IEEE/ACM International Symposium on Computer Architecture (ISCA)*, 2022.

[41] Y. Zhang, H. Ren, and B. Khailany, "GRANNITE: Graph Neural Network Inference for Transferable Power Estimation," in *ACM/IEEE Design Automation Conference (DAC)*, 2020.

[42] W. R. Davis, P. Franzon, L. Francisco, B. Huggins, and R. Jain, "Fast and Accurate PPA Modeling with Transfer Learning," in *IEEE/ACM International Conference On Computer Aided Design (ICCAD)*, 2021.

[43] D. Lee, L. K. John, and A. Gerstlauer, "Dynamic Power and Performance Back-Annotation for Fast and Accurate Functional Hardware Simulation," in *Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2015.

[44] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," in *International Conference on Learning Representations (ICLR)*, 2017.

[45] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Liò, and Y. Bengio, "Graph attention networks," in *International Conference on Learning Representations (ICLR)*, 2018.

[46] Y. Shi, Z. Huang, S. Feng, H. Zhong, W. Wang, and Y. Sun, "Masked Label Prediction: Unified Message Passing Model for Semi-Supervised Classification," in *International Joint Conference on Artificial Intelligence (IJCAI)*, 2021.

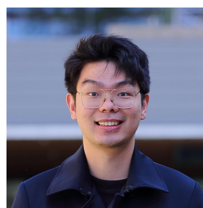
[47] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," in *International Conference on Neural Information Processing Systems (NeurIPS)*, 2017.

[48] L.-N. Pouchet, "Polybench: The polyhedral benchmark suite," 2012. [Online]. Available: http://web.cs.ucla.edu/ pouchet/software/polybench/

[49] B. Reagen, R. Adolf, Y. S. Shao, G.-Y. Wei, and D. Brooks, "MachSuite: Benchmarks for accelerator design and customized architectures," in *IEEE International Symposium on Workload Characterization (IISWC)*, 2014.

[50] Y. Hara, H. Tomiyama, S. Honda, H. Takada, and K. Ishii, "CHStone: A benchmark program suite for practical C-based high-level synthesis," in *IEEE International Symposium on Circuits and Systems (ISCAS)*, 2008.

[51] O. Sener and S. Savarese, "Active Learning for Convolutional Neural Networks: A Core-Set Approach," in *International Conference on Learning Representations (ICLR)*, 2018.



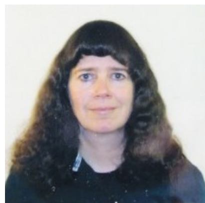
Zhigang Wei received the B.Eng. degree in Electronics and Communication Engineering from City University of Hong Kong in 2017, and Ph.D. degree in Electrical and Computer Engineering from the University of Texas at Austin in 2025. His research interests are in the areas of Machine Learning based power modeling of FPGA High-Level-Synthesis designs. He is currently working as Software Engineer in Power Rail Analysis at Cadence.

Authorized licensed use limited to: University of Texas at Austin. Downloaded on March 05,2026 at 21:28:52 UTC from IEEE Xplore. Restrictions apply.
© 2025 IEEE. All rights reserved, including rights for text and data mining and training of artificial intelligence and similar technologies. Personal use is permitted,
but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.

This article has been accepted for publication in IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems. This is the author's version which has not been fully edi
content may change prior to final publication. Citation information: DOI 10.1109/TCAD.2025.3646555

Allison Seigler is a PhD student in Electrical and Computer Engineering at the University of Texas at Austin. She received her B.Sc. degree in Computer Engineering from the University of Wisconsin-Madison. Her research interests include resilience, performance, and power modelling of existing and novel hardware accelerator architectures, including GPUs and FPGA-based accelerator designs.

Sean Lowe is a graduate student in Computer Engineering at Arizona State University, where he also completed his bachelor's degree in Computer Systems Engineering. His work focuses on applying machine learning and AI to hardware design and verification, including AI-driven testbench generation and intelligent automation in semiconductor flows. He previously worked at Amkor Technology, where he applied AI and machine learning to support advanced semiconductor manufacturing. His research interests include heterogeneous computing, reconfigurable systems, and AI-assisted EDA.



Emily Shriver is a Research Scientist in systems design and architecture. She conducts research on power and performance analysis, modeling and simulation techniques with expertise spanning a spectrum of abstraction levels including circuits, RTL, emulation, micro-architecture, networks and system level platforms including large-scale distributed systems. As a Principal Engineer and Research Scientist at Intel Labs, her research developed performance analysis and modeling techniques for large scale data center networks enabling HW/SW co-design, multiple power modeling techniques for emulation and research on creating privacy preserving proxies. She has co-authored refereed journal and conference and journal publications, two best paper nominations, been an invited technical panelist at DAC, and served on several technical program committees including ISCA, HPCA, and DAC.

Aman Arora (Member, IEEE) received the B.Tech. degree in electronics and communications engineering from National Institute of Technology Kurukshetra, Haryana, India in 2007, and the M.S. and Ph.D. degrees in electrical and computer engineering from The University of Texas at Austin, Texas, USA in 2012 and 2023 respectively. He is currently an assistant professor at Arizona State University, Arizona, USA, where he leads a research laboratory working on developing computer systems that can efficiently process modern workloads such as Machine Learning (ML), with a special focus on reconfigurable systems. He has over 10 years of experience in the semiconductor industry in design, verification, testing and architecture roles.

Dr. Lizy Kurian John holds the Truchard Foundation Chair in Engineering in the Department of Electrical & Computer Engineering at The University of Texas at Austin. Her research is in the areas of computer architecture, multicore processors, memory systems, performance evaluation and benchmarking, workload characterization, and reconfigurable computing. Prof. John's research has been supported by the National Science Foundation, Semiconductor Research Consortium (SRC), DARPA, Lockheed Martin, AMD, ARM, Oracle, Huawei, IBM, Intel, Motorola, Freescale, Dell, Samsung, Texas Instruments, etc. She is recipient of NSF CAREER award, UT Austin Engineering Foundation Faculty Award, Halliburton, Brown and Root Engineering Foundation Young Faculty Award, University of Texas Alumni Association Teaching Award, The Pennsylvania State University Outstanding Engineering Alumnus Award, etc.

Lizy John holds 20 U. S. patents and has published four books, 17 book chapters, 300+ refereed journal and conference publications, and more than 50 workshop papers. Prof. John served as the Editor-in-Chief of IEEE Micro, and has served in the editorial boards of IEEE Transactions on Computers, IEEE Transactions on VLSI, IEEE Transactions on Sustainable Computing, IEEE Computer Architecture Letters, ACM Transactions on Architectures and Code Optimization. She is an IEEE Fellow, ACM Fellow, National Academy of Inventors (NAI) Fellow, IEEE Golden Core Member and Fellow of the National Academy of Inventors.

Authorized licensed use limited to: University of Texas at Austin. Downloaded on March 05,2026 at 21:28:52 UTC from IEEE Xplore. Restrictions apply.
© 2025 IEEE. All rights reserved, including rights for text and data mining and training of artificial intelligence and similar technologies. Personal use is permitted,
but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.