ATAPP: Architecture and Technology Aware Power Predictor for Unseen FPGAs

Zhigang Wei*, Aman Arora†, Emily Shriver, Lizy K. John*
*The University of Texas at Austin, †Arizona State University
zw5259@utexas.edu, aman.kbm@asu.edu, ljohn@ece.utexas.edu

***Abstract*—Machine Learning has been successfully adopted to estimate power consumption for FPGA designs using features from the early stages of High-Level synthesis. However, existing ML-based power models work only for the same FPGA the model is trained on, and do not generalize well for unseen or futuristic FPGAs due to the lack of consideration of architectural and technology features, which are highly relevant to the power consumption. In addition, state-of-the-art analytical models take hours to synthesize and simulate hardware designs to perform accurate power analysis. It leads to a long turnaround time and inefficiency for researchers who want to explore the effects of the FPGA architecture on power. In order to tackle the problem, we proposed ATAPP, a novel Graph Neural Network-based power model where both design switching activities and architecture features are encoded to make power prediction. The features of the FPGA architecture are extracted with RapidWright at the tile level of the layout, and the embeddings are generated using positional encoding. The design features are composed of switching activities at the intermediate representation operator level and the data flow of the design in a graph. With the two representations, ATAPP is able to predict the average dynamic power with a new design overlay on an unseen FPGA. Our experiments show that ATAPP demonstrates an average error of 13.09% with unseen FPGAs and designs, while the best prior produces more than 40% error and XPE produces more than 200%.**

I. Introduction

Power efficiency has emerged as one of the first-order constraints for hardware systems such as field-programmable gate arrays (FPGAs), and both the FPGA architecture and design optimization with regard to power efficiency usually necessitate knowledge of power consumption. However, the power evaluation flow for FPGA designs induces large overheads of design turnaround time, and the problem becomes more serious when researchers want to explore the architecture of an FPGA considering power metrics in addition to performance metrics. In general, accurate FPGA power estimation requires the signal activities of critical components and I/O ports to be obtained via vector-based gate-level simulation and a set of physical component measurements to be obtained through the Register-Transfer-Level(RTL)-based FPGA implementation flow, including synthesis and implementation. After these steps, analytical models can be used to infer power consumption. However, all the above steps need to be refined and repeated once the FPGA architecture changes, since the resource types or counts and their layout on the FPGA fabric can greatly affect the implementation phase even with the same RTL design. The technology node can further influence all the mentioned phases and the analytical model for power evaluation. The cycle-accurate gate-level simulation and design implementation flow with different FPGA architecture result in large runtimes. Overall, in a power-oriented hardware optimization loop, designers repeatedly perform the above power evaluation steps until power closure is achieved, which incurs long development time and high labor cost.

To overcome the challenge of post-implementation power evaluation in FPGA, previous works have used machine learning (ML) techniques [1], [2] to bypass bottleneck phases including synthesis and implementation. ML models are trained with features extracted from the High Level Synthesis (HLS) [3] phase and labels from simulation-based power consumption. The models mentioned above are designed to predict power with HLS features on one specific FPGA, however, these models cannot be used in power prediction on unseen FPGAs. We compare the power over 4,000 designs on 2 different FPGAs and the relative power difference (calculated as $\frac{P_2-P_1}{P_1}$) is shown in the right of Fig. 1. For a specific view of the power comparison as illustrated in the left of Fig. 1, we pick 6 designs and compare the power consumption of these 6 designs on 2 different FPGAs (zu9eg as FPGA1 and 7v585t as FPGA2). It is observed that there is a significant power difference between FPGA1 and FPGA2. Therefore, power models, which can make accurate prediction on FPGA1, cannot make correct prediction on FPGA2 (a new unseen FPGA) due to the features dependent solely on the designs and lack of consideration of FPGA architecture features.

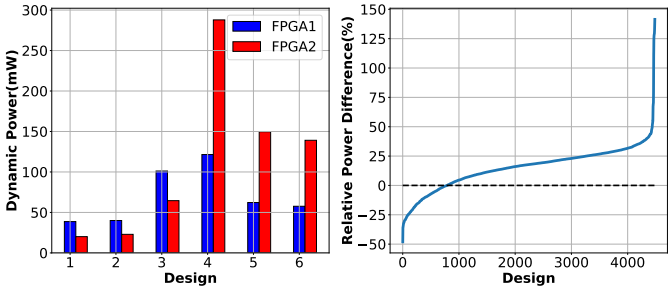

Fig. 1. Changes in power from one FPGA to another. Power may increase or decrease as shown (left) for example designs; Distribution of the relative power difference on two different FPGAs based on over 4000 designs (right)

We summarize and compare the prior analytical power model and existing ML-based power models in Table I. There are integrated power estimation tools inside AMD/Xilinx Vivado and it supports the power estimation for all the Xilinx FPGAs, however, it lacks the flexibility to estimate the power in different design cycles. It takes a long time to synthesize and simulate designs before accurate power estimation. The AMD Power Estimator(XPE) is an analytical power model that supports flexible estimation in different design phases. It supports all AMD/Xilinx devices, but accuracy is too low with early-stage HLS results [4]. Both Vivado and XPE only support power estimation on existing devices, and researchers are not allowed to manipulate the FPGA architectures. Verilog-to-Routing (VTR) [5] is an open source tool used for the exploration of FPGA architecture. Although manipulation of the FPGA architecture is allowed, it takes even longer than Vivado to synthesize the design and estimate the power. PowerGear [1] is state-of-the-art ML-based power model, it can make prediction accurately on the FPGA seen in the training, it performs poorly for unseen FPGAs. In order to predict power for the unseen FPGA with PowerGear, it requires a complete dataset generation and training which restrict the usage of PowerGear on varities of FPGAs. Therefore, fast and accurate power evaluation on unseen FPGAs remains unsolved.

TABLE I
A COMPARISON OF ATAPP WITH THE EXISTING WORK ON POWER PREDICTION. ●: FEATURE SUPPORTED; ○: FEATURE UNSUPPORTED; ◐: FEATURE PARTIALLY SUPPORTED.

Features	Vivado	XPE [4]	VTR [5]	PowerGear [1]	ATAPP
High accuracy	●	○	◐	◐	●
Time efficient	○	●	○	◐	●
No logic synthesis	○	◐	○	●	●
No implementation	○	◐	○	●	●
Unseen FPGA Prediction	○	○	●	○	●

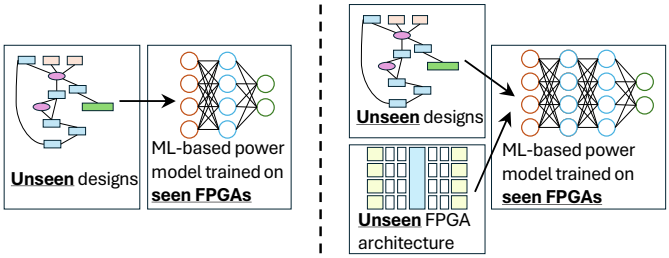

Fig. 2. Prior power models predict power of unseen designs on seen FPGAs (left) vs. ATAPP which predicts power of new designs on new FPGAs (right)

We propose ATAPP, an FPGA architecture and technology-aware power predictor that combines both the features of the FPGA architecture and the switching activities of the designs to predict the average dynamic power consumption. Unlike the existing ML-based power model, ATAPP makes predictions based on two parts: FPGA architecture representation and design representation as shown in Fig. 2. The architecture representation includes information such as resource type and counts, technology parameters, resource position in the FPGA fabric, and voltage levels. They are encoded into finite length of vectors so that they can be digested by a multi-layer perceptron (MLP) model to generate architecture embeddings. For the design representation, we represent the HLS design as a graph which is generated with Intermediate Representation (IR) codes and Finite State Machine Datapath (FSMD) model produced by HLS compilation. We then extract switching activities by simulating the IR codes and put them on the associated edges of the graph. A GNN encoder, which is adapted from UniMP [6], is applied to the HLS designs to generate design embeddings. Both architecture and design embeddings are concatenated and fed into another MLP for a regression task. ATAPP, to our knowledge, is the first work to predict power across FPGAs based on both FPGA architecture and HLS designs. In this paper, we target Xilinx FPGAs as an example, but our approach is extendable to other FPGA vendors and academic FPGA architecture research. Our contributions are summarized as follows:

- We propose a GNN-based model ATAPP, an architecture-aware and technology-aware power prediction model. It is trained on seen FPGAs and seen designs (circuits) but can make power prediction for both unseen FPGA architectures and designs.
- We apply positional encoding to efficiently encode the FPGA architecture features at the tile level and generate FPGA architecture embeddings with a MLP model.
- A GNN encoder based on UniMP layer is developed to generate graph embeddings aware of switching activities, generated with IR codes and FSMD model.
- The experimental results show that ATAPP can predict dynamic power with a 13.09% error on average.

II. RELATED WORK

Machine Learning for EDA. Machine Learning (ML) algorithms have gained popularity in the Electronic Design Automation (EDA) domain due to their extremely high efficiency, high quality [7], 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), [1], [2], [8]–[15], logic synthesis [16], [17], and placement and routing in physical design [18]–[22]. As [7] 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 [11], [15], [23]; (2) Cross-platform performance prediction such as performance prediction for new FPGA platforms and performance prediction for new applications through the execution on CPUs [13], [24]; (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 [10], [25]; (4) Improving optimization algorithms where ML models are used to substitute traditional algorithms for hyper-parameter or configuration selection [26], [27]. Our work focuses on power prediction for HLS designs on unseen FPGA platforms.

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 cycle to perform power estimation including architecture-level power prediction [28]–[30], RTL stage power modeling [31]–[39] and HLS stage power estimation [1], [2], [13]–[15], [40], [41]. Compared to architecture-level power estimation, HLS designs provide a closer look of the hardware designs, and thus more accurate power estimation can be produced. Althgouh 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 [1], 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 [13] 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. All of the above-mentioned works perform well for the same FPGA they are trained with, but their models are not aware of FPGA architectures restricting their model usage on single FPGA. Our work aims to provide an FPGA architecture-aware power model so that it can predict power for HLS designs on unseen FPGAs.

III. PROBLEM FORMULATION

In this work, we aim to predict the power for an unseen design implemented on an unseen FPGA using an ML technique. For this matter, we need to first define and solve the following problems step by step:

Problem 1: Generate the FPGA architecture representation. The architecture features should be both representative of the FPGA characteristics and compatible with machine learning algorithms. The key challenge is to find a good architecture representation ensuring sufficient granularity to reflect relevant details while avoiding unnecessary complexity that could hinder model generalization. Therefore, we split the FPGA architecture representation into several categories that can highly impact power consumption for the device denoted as d: resource counts and types β such as LUTs, flip-flops, DSP slices and BRAM, technology parameter and voltage levels γ and the layout map δ of resources and switch boxes. We consider a device is determined by these three factors $d(\beta,\gamma,\delta)$.

Problem 2: Generate the design (circuit) representation with switching activity It is a common way to use a graph to represent a HLS design [1], [11]. The graph should sufficiently represent the hardware dataflow as well as the switching behavior of the entire design. Therefore, for the graph $G=(V,E)$ where V and E represent the node and edge set, respectively, every node in the graph can be used to represent the operator and every edge in the graph can be used to represent the switching behavior on that path.

Problem 3: Build the prediction model with both the design (circuit) and the unseen FPGA architecture. Let g be the HLS generated codes with switching activities α on a FPGA device d with resource and interconnect representation β, technology parameter γ and layout map δ. Let P be the the ground-truth power generated by the vendor FPGA implementation and simulation tool:

$$P=F(g(\alpha),d(\beta,\gamma,\delta)) \tag{1}$$

The goal is to find a hypothesis H that approximates the results of function F for any given HLS generated codes g with any switching activities α on any FPGA device d that can be defined by the resource and interconnect features r, technology parameter t and layout map δ:

$$\min_{H} Loss(F(g(\alpha),d(\beta,\gamma,\delta)),H(g(\alpha),d(\beta,\gamma,\delta))) \tag{2}$$

IV. Methodology

The objective of our study is to generate a general ML-based model which can make power predictions based on the FPGA architecture and design representation. As discussed in Section III, our solution mainly includes an effective FPGA architecture and representation of the HLS design, as well as a model construction. ATAPP is trained with a set of FPGA architecture, design overlay on that FPGA and the associated ground-truth power. It can then make prediction with unseen FPGA architecture and design representation. We discuss how the architecture representation is encoded in details in Section IV-A and how the design representation is generated in Section IV-B. Section IV-C discusses the detailed view of the model construction and how the architecture and design embeddings are produced. Based on the definition of architecture and design features, we generate our own dataset for our studies.

A. Architecture representation

It is important to capture features of FPGA architecture to make prediction for unseen FPGAs, but it is not straightforward to feed power-related FPGA architecture features directly to an ML-based model since not all features are numeric. Representing an FPGA architecture, however, in a numerical way poses several challenges due to the inherent complexity, flexibility and highly configurable nature of FPGAs. These challenges arise from the spatial structure of the architecture, for example, how to arrange the basic unit on the fabric matters a lot to the implementation and hence the power consumption of the same HLS design but to make these arrangements consumable by an ML model requires extra effort.

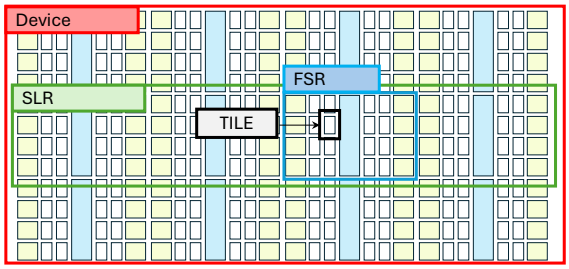

Fig. 3. FPGA Architecture Terminology: Super Logic Region (SLR), Fabric Sub Region (FSR)

As discussed in Section III, we define the power prediction problem scoped by resource and interconnect representation, technology parameters and layout map. The resource representation and technology parameters can be expressed as numeric values. Therefore, in order to explore the effect of these features on power prediction, we further define the FPGA type within the AMD/Xilinx UltraScale+, UltraScale and 7Series families. The resource count is one of the important features for the capability of an FPGA device. Therefore, the number of these resources, E_{res} (DSP, LUT, FF and BRAM), are included as key features. Even though the UltraScale and UltraScale+ FPGA devices use different DSP structure from 7Series, we do not observe any obvious number of utilized DSP difference between the devices when implementing the same HLS design. All the FPGA devices mentioned use the same structure of other units, therefore, we do not go deeper to reach the gate-level implementation the basic components. Besides the available resource counts, technology parameters E_t, including technology node and operational voltage, are also critical to the power dissipation.

When it comes to spatial features of the FPGA architecture, we need to first define which abstraction level to reach. In Xilinx FPGAs, there are six major levels of hierarchy - the entire device all the way down to building blocks. They are **Device, Super Logic Region (SLR), Fabric Sub Region (FSR), Tile**, Site and Basic Element of Logic (BEL) [42]. The first four hierarchies can be seen in Fig. 3. The device is at the highest level of Xilinx architecture and it is composed of replicated FSRs. SLRs are present in certain devices and each SLR contains a 2D array of FSRs. FSR is a 2D array of tiles in the fabric. Each tile is an instance composed of multiple sites and each tile has a unique name with a coordinate suffix. Not all tiles contain sites (there exist NULL and empty tiles in Xilinx terminology), but those that do can have more than one. A site is referred to as a group of related elements and their connectivity. Similar to a tile, each site is associated with its own coordinate grid and there is a chance that two sites share the same grid space. The site type includes SLICEL and SLICEM which are the most common site types and are the basic configurable logic building blocks (CLBs) that contain LUTs and FFs. At the lowest level, the atomic unit is a BEL. BELs are the smallest, indivisible, and representable component in the fabric of an FPGA. In order to collect the aforementioned information, we utilize **RapidWright** [42], which is an open source Java framework that provides accurate device model views of all Vivado-supported Xilinx devices including 7Series, UltraScale and UltraScale+. The encoding flow of the FPGA architecture includes several steps and is summarized in Fig. 4. We introduce the process in detail below.

❶ **FSR extraction.** Since each device is composed of several similar FSRs, modeling every FSR separately is not required to

TABLE II
FEATURES USED IN OUR MODEL

Feature category	Format	Details
Architecture	360×1×4 matrix with PE: AE	Resource types and the arrangement of resources on fabric
Design	Graph data with node and edge attributes: $G=(V,E)$ and metadata	DFG recovered from IR codes, FSMD model and post-HLS reports including achieved clock period, overall latency and resource utilization
Architecture Metadata	A vector of size 21: $E_{res}\|\|E_t$	Available resource counts on the device, # of rows and columns of FSR, # of valid rows and columns of tiles, technology node (in nanometers), voltage levels on the device (V_{ccint},V_{ccbram},$etc.$), the size of FPGA

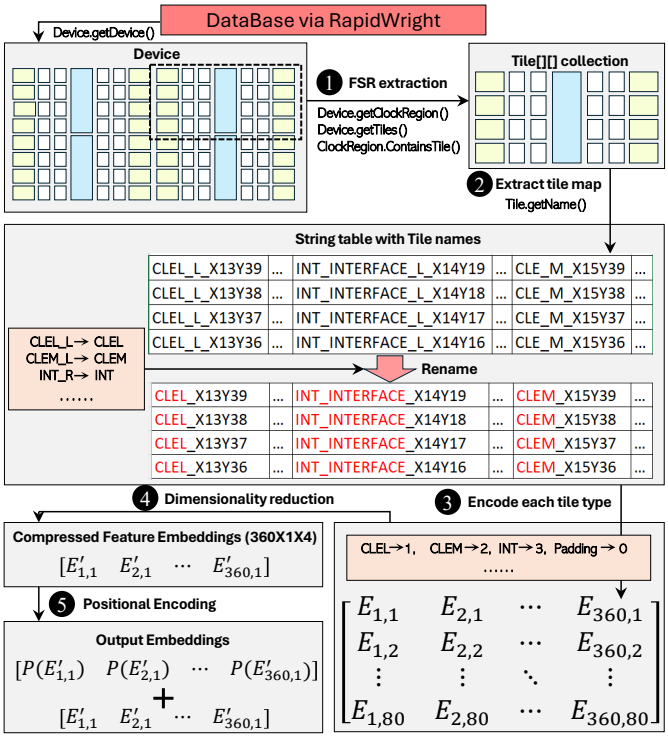

Fig. 4. FPGA Architecture Encoding

create a representation of the entire FPGA architecture. Therefore, we choose to extract tile information from one of the FSRs. Specifically, `Device.getDevice()` is firstly used to choose the FPGA we want to extract, `Device.getClockRegion()` is used to extract the clock region. The FPGA tile collection is extracted with `Device.getTiles()`. We check all the tiles and reduce the tile collection down to the selected FSR with `ClockRegion.containsTile()`. Since every device may have different number of FSRs, the number of rows and columns of FSRs should be also included. They are included in the metadata and appended to the final embeddings.

❷ **Extract tile map.** Once we obtain the tile collection of the selected FSR, the next step is to transfer these tile objects into a string table that is easy to understand and parse. `Tile.getName()` is used to find the name of each tile. Every tile name is composed of the resource type as its prefix and the grid location as its suffix. For example, `INT_X14Y519` refers to a switch box tile located at grid (14, 519). Before we advance to the next step to encode the string table, we need to reduce the resource space first, since the total number of unique tile types can be too large and not all of them are frequently used in the device. The tile type counts range from 180 up to around 360, and to encode all of them incurs too much sparsity when generating the embedding and training may become difficult as a consequence. Therefore, we select the tile that includes a specific resource type and remove others from our string table. Seven tile types are selected and they include `CLEL, CLEM, INT, INT_INTERFACE, DSP, BRAM, BRK` where `CLEL, CLEM` refer to the tile containing CLB, `INT, INT_INTERFACE` refer to the tile containing switch box and `BRK` refers to a break tile that disallows any crossing. However, each tile name cannot exactly fall into the seven categories due to the inconsistent naming nature in the FPGA series, for example, `CLBLL_L` is used in 7Series FPGAs but `CLEL_L` is used in others. Therefore, we check the name of each tile and determine where it belongs: ① All tiles with name containing `CLEL` belong to `CLEL`. ② `CLBLL` is encoded as `CLEL`. `CLBLM` and `CLE_M` are encoded as `CLEM`. ③ `NULL` tiles encoded as one of `DSP`, `BRAM` and `BRK` based on its column. ④ `INT` and `INT_INTERFACE` are encoded seperately. With the above rules, more than 97% tiles from FSR can be encoded into our selected seven tile types and almost all the grids are occupied by them.

❸ **Encode each tile type.** Once we complete the renaming of the string table. We should encode the table into a matrix of embeddings. We notice that each grid location, which is identified by the suffix `_X#Y#` of each string, contain no more than 4 tiles. Therefore, each embedding should be a size 4 vector. For the grid which has less than 4 tiles, pad the embedding with a dummy 0. An example tile `[CLEM_X31Y815, INT_X31Y815, CLEL_X31Y815]` can be encoded as a single embedding `[2, 3, 1, 0]`. Considering that the grid size for FSR is not identical across different devices, we define the matrix size to be 360 columns × 80 rows which is larger than the largest grid size of our selected FPGAs (350×77). We add zero padding to the matrices for the empty place. We end up generating a 3d matrix with size of 360×80×4 (#columns×#rows×#channels).

❹ **Dimensionality reduction.** Xilinx leverages a columnar architectural approach to tile layout. That is, with a few exceptions, all tiles within a column are of the same type but tiles occupying the same row are typically different types. Therefore we can further compress the matrix size from 360×80×4 into 360×1×4 since each row is simply a replicate of the first row. Extra metadata is needed to indicate the number of rows and columns with valid tiles (not padding zero). The embeddings eventually becomes a 360×1×4 with zero-padding at the end.

❺ **Positional Encoding.** Since the sequence to place the tiles on fabric is critical to solving the power prediction, we use *Positional Encoding (PE)* to encode the resource layout of FPGA devices. *PE* is a key concept in transformer models [43], designed to provide information about the order of sequence or spatial elements. Since transformer architectures are inherently permutation-invariant, they lack an innate sense of sequence order, which is critical for tasks involving sequential/spatial data. Hence, PE is added to the input embeddings to include order information. This encoding can be learned or predefined; a popular approach uses sinusoidal functions with different frequencies to generate unique values for each position in the sequence. This method ensures that the positional encoding generalizes to sequences of varying lengths and maintains properties conducive to understanding relative and absolute positions. A typical

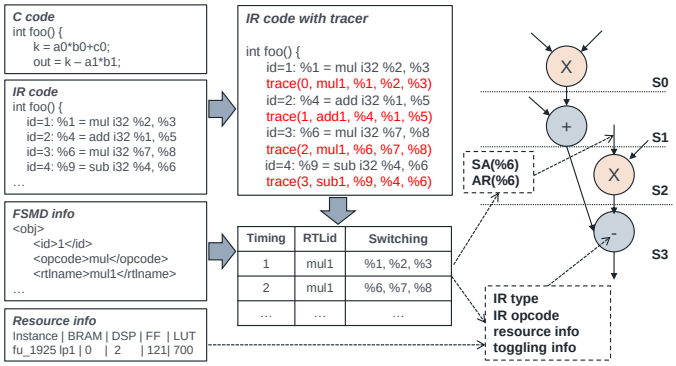

Fig. 5. Design representation generation flow

PE equation is defined as:

$$PE_{(pos,2i)} = sin(pos/10000^{2i/d})$$
$$PE_{(pos,2i+1)} = cos(pos/10000^{2i/d})$$

where *pos* is the position of the input embedding in the sequence. *i* is the index of the dimension in the embedding vector and *d* is the dimensionality of the embedding which is 4 in our case. The positional encoding is calculated and directly added to the input embeddings to generate output architecture embeddings AE.

B. Design representation

The power-aware design representation is mainly 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. Therefore, in order to correctly represent the two factors, IR codes with FSMD model generated by HLS tools are chosen. Similarly in [1], [41], the design generation flow used in ATAPP is summarized in Fig. 5. Three main files generated from HLS tools are used: IR codes, FSMD model and resource utilization for each operator. In order to trace the toggling behavior of each operator, a trace function is inserted after every important IR instruction. With the IR-level simulation, the behavior of each IR operator can be recorded. With the FSMD model, each IR operator can be mapped to a specific RTL hardware operator, the IR operators sharing the same hardware resources can be merged and a graph close to the hardware can be generated. For the generated 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 a IR operator with attributes that include opcode, opcode type, 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 $AR_{i,j}$:

$$SA_{i,j} = \frac{\sum HD(v_i(k), v_i(k-1))}{L}, \quad AR_{i,j} = \frac{N}{L} \tag{3}$$

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.

C. Predictive Model

The features and format of the features used in our model are listed in Table II. The detailed structure of ATAPP is shown in Fig. 6. It is made up of three major neural networks: a GNN encoder to generate graph embeddings for design representation, a MLP model used to generate architecture embeddings, and another MLP model used to perform power regression task.

GNN encoder: Edge attributes, including switching features, are essential to perform power prediction. However, both GCN [44] and GAT [45] overlook the edge embeddings. Although PowerGear [1] propose 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 [6], inspired by Transformer [43] used a different aggregation mechanism on each edge. It builds attention coefficients $\alpha_{i,j}$ with both edge and node attributes in every UniMP layer:

$$\alpha_{i,j}^{(l)} = softmax\left(\frac{(W_1^{(l)} h_i^{(l)})^T (W_2^{(l)} h_j^{(l)} + W_3 e_{i,j})}{\sqrt{D}}\right) \tag{4}$$

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 destination j to the source i:

$$h_i^{(l+1)} = \sum_{j \in N(i)} \alpha_{i,j}^{(l)} \left(W_4 v_j^{(l)} + e_{i,j}\right) \tag{5}$$

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 and further concatenate the graph embedding with the design metadata h_M extracted from post-HLS reports and it includes achieved clock period and latency. The aggregation and concatenation are formularized as follows:

$$h'_G = \sum_{l \in L} \sum_{v \in V} h_v^{(l)}, \quad h_G = h'_G || h_M \tag{6}$$

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.

MLP encoder and MLP decoder: we use the MLP to encode the architecture because the feature space is relatively simple. The MLP encoder is made up of 2 hidden layers. The MLP decoder is used to perform power regression task and is composed of 2 hidden layers. Every MLP hidden layer is followed by ReLU activation. The power estimation P_{est} is calculated as follows:

$$P_{est} = MLP_1(h_G || h_A), \quad h_A = MLP_2(h_{AE} || E_t || E_{res}) \tag{7}$$

V. Experimental Results

A. Experiment Setup

Our dataset includes 5,391 designs for each FPGA (over 43,000 in total). These designs are generated from benchmark kernels of intermediate complexity that can be used as building blocks of larger applications. Specifically, we selected kernels from the widely used MachSuite [46], CHStone [47], and Polybench [48] benchmark. They include kernels with different computation intensities including linear algebra operations on matrices and vectors, data mining (correlation and covariance), stencil operations, encryption, and a dynamic programming application as shown in Table III. We automate the HLS design generation using HLSFactory [49], which is a framework designed specifically for the HLS dataset generation.

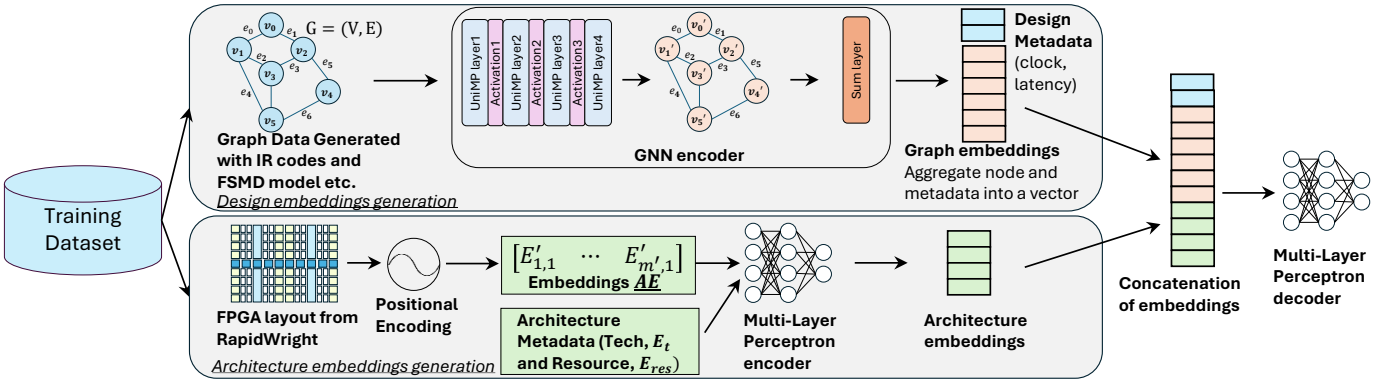

Fig. 6. Detail structure of ATAPP composed of *design embeddings generation* using GNN encoder and *architecture embeddings generation* using MLP; the power regression is conducted with a MLP

Since it does not support the simulated power evaluation on the implemented designs, we further run the simulation on the generated designs to collect ground-truth power for each FPGA.

AMD/Xilinx Vivado HLS 2018.3 and Vivado 2018.3 are used to synthesize and simulate the design to collect the design features and ground-truth power. We selected 8 different FPGA devices from AMD/Xilinx for our experiments as shown in Table IV, and these 8 devices cover all the families supported by current Vivado tool chain. The FPGA architecture representation is processed with the raw data from RapidWright as discussed in section IV-A. Our model is implemented and trained using PyTorch. The dataset of designs for each FPGA is split into 85% for training, 15% are reserved for testing purpose. The specific split of FPGA is discussed together with the results in the evaluation section. We use the Adam optimizer and a learning rate of 0.001 during the training.

We use three metrics to evaluate the prediction accuracy: correlation coefficient (R), Mean Absolute Error Percentage (MAPE) and Root Relative Square Error (RRSE), which are defined as following:

$$\text{MAPE} = \frac{1}{n}\sum_{i=1}^{n}\left|\frac{y_i - \hat{y}_i}{y_i}\right| \times 100\%, \quad \text{RRSE} = \sqrt{\frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{n}(y_i - \bar{y})^2}} \tag{8}$$

where n refers to the number of samples, 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. These metrics bring a comprehensive and fair evaluation of ML models from three aspects, where higher correlation R, lower MAPE and RRSE indicate better model performance and accuracy.

TABLE III
BENCHMARK KERNELS USED FOR THE DESIGN DATASET PER FPGA

Benchmark suite	Kernel application	# Designs
Polybench	atax, bicg, gemm, gesummv, k2mm, k3mm, mvt, syrk, syr2k	4779
Machsuite	spmv_crs, stencil3d, stencil2d	376
CHStone	aes, gsm, sha	281

B. Baseline Solutions

While there exists no prior architecture-based power model with ML technique to our best knowledge, AMD/Xilinx Power Estimator(XPE) [4] and PowerGear [1] are selected to compare against our work. XPE is a spreadsheet based tool that estimates the power consumption of the design at any stage during the design cycle. It is typically used in the pre-design and pre-implementation stages. It accepts design information through simple design wizards, analyzes

TABLE IV
OVERVIEW OF CHARACTERISTICS OF FPGAS USED IN EXPERIMENTS

#	Series	Device	Tech	#LUTs	#BRAMs	#DSPs
1	UltraScale+	zu9eg	16nm	274,080	912	2,520
2	UltraScale+	vu3p	16nm	394,080	720	2,280
3	UltraScale+	au25p	16nm	141,000	300	1,200
4	UltraScale	vu440	20nm	2,532,000	2,520	2,880
5	UltraScale	ku115	20nm	663,360	2,160	5,520
6	7Series	7a200t	28nm	134,600	365	740
7	7Series	7k480t	28nm	298,600	955	1,920
8	7Series	7v585t	28nm	364,200	795	1,260

them and provides detailed power and thermal information. However, XPE is designed to estimate worst-case power and therefore the estimated power tends to be larger than the ground-truth power, which is generated with Vivado synthesis and simulation tools as described in Section V-A. PowerGear, on the other hand, is a state-of-the-art GNN-based power model for HLS designs. It constructs graph samples for designs with HLS-generated reports and switching activities from simulation. Their objective is to generate the model for single FPGA, therefore, the model performs poorly when directly used in other FPGAs. Before we delve into our major experimental results, some preliminary experiments are conducted with these baseline solutions.

TABLE V
XPE ESTIMATION ERROR COMPARED TO VIVADO IN DIFFERENT DESIGN PHASES FOR ULTRASCALE VIRTEX VU440
P_{syn},P_{impl}: POWER ESTIMATED BY VIVADO AFTER SYN AND IMPL
P'_{syn},P'_{impl}: POWER ESTIMATED BY XPE AFTER SYN AND IMPL

{Act. Est.}	$\{P_{syn}, P'_{syn}\}$	$\{P_{impl}, P'_{impl}\}$
MAPE	26.26%	5.03%
{Act. Est.}	$\{P_{impl}, P'_{syn}\}$	$\{P_{impl}, P'_{hls}\}$
MAPE	37.78%	390.07%

XPE can estimate the power from any design cycle, but the accuracy of XPE is largely dependent on how much information can be entered into it. XPE considers the design resource usage, toggle rates and many factors which it combines with the device models to calculate the estimated power distribution. It accepts two primary sets of inputs: ① device usage, component configuration, clock, enable, and toggle rates, and ② device data models which are already integrated into the tool. With the specification of designs extracted from Vivado, XPE results still deviate from the Vivado results. A preliminary experiment is conducted: we simulate the implemented design and generate the switching activity in a vector file (`.saif`). The vectors are then used in Vivado power estimator on both

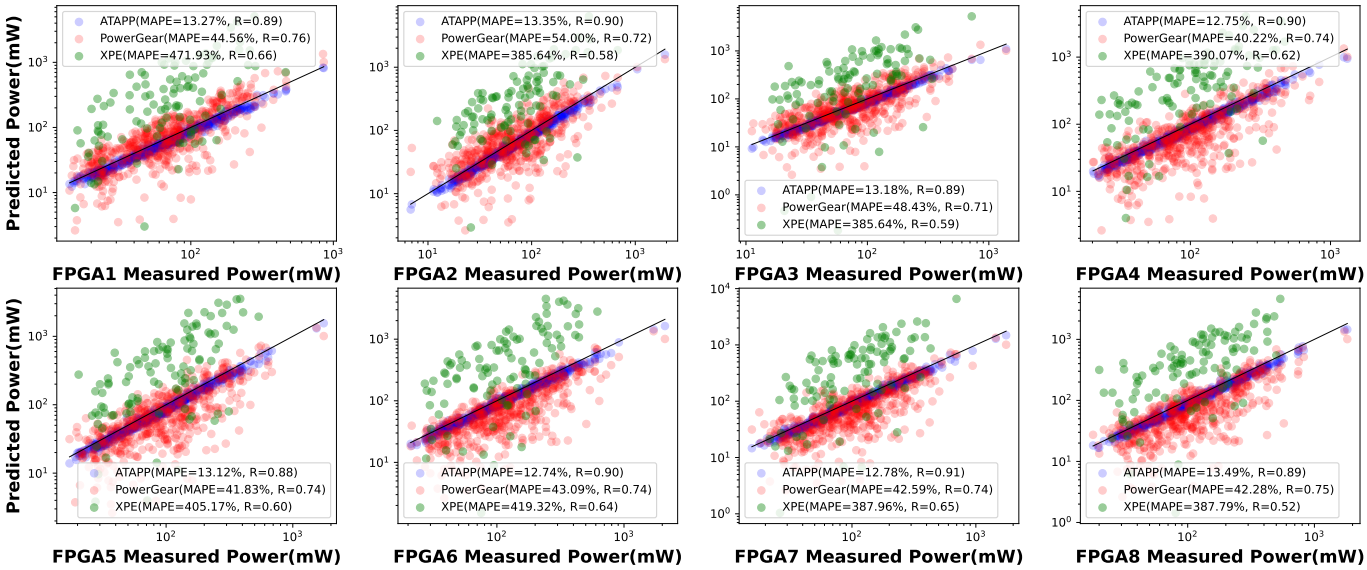

Fig. 7. Prediction vs ground-truth for dynamic power consumption of all designs. The black line in the middle of each figure indicates **zero error** (i.e. predicted power equals to the ground-truth power). **ATAPP**: **leave-one-out** strategy is used, the model is tested with the designs on one FPGA and trained with the designs on all other FPGAs. 8 models are trained and tested independently. **PowerGear**: train with the designs on FPGA8 and test with the designs on FPGA1-FPGA7; train with the designs on FPGA1 and test with the designs on FPGA8. Blue - ATAPP; Red - PowerGear; Green - Vivado XPE.

synthesized and implemented design to generate dynamic power P_{syn}, P_{impl} respectively, the settings are also extracted as XPE compatible files `syn.xpe, impl.xpe`. These files are then used in XPE to estimate the core dynamic power P'_{syn},P'_{impl}. Vivado does not support power report at HLS stage and there is no way to extract `.xpe` file right after HLS. We use the HLS estimated resource and default toggling rate 12.5% to estimate power with XPE, the core dynamic power is denoted as P'_{hls}. The XPE error is concluded in Table V. The first row $\{P_{syn}, P'_{syn}\}$ and $\{P_{impl}, P'_{impl}\}$ indicate that even with the same configuration, XPE still deviates from the actual value. Compared to the predicted power of the target P_{impl}, as the amount of information accepted by XPE decreases, the error increases significantly if no appropriate switching activity is entered.

PowerGear trains GNN models to predict the post-implementation dynamic power. We use simulated power as ground truth power rather than the measured power in the original paper. The GNN model takes graph representation of HLS designs as input to make prediction. Although it performs quite well on the same FPGA it is trained on (MAPE=5.08%), since it is designed for prediction for unseen FPGA power the error increases greatly when the FPGA changes (MAPE=44.63%). The error comes from the difference in FPGA architectures between unseen FPGAs and trained FPGAs, and the model is not able to adapt to unseen FPGAs without consideration of FPGA architectures since PowerGear is not an architecture-aware power model, therefore, ATAPP is needed.

C. Model Evaluation

Fig. 7 and Table VI show the comparison of ATAPP over PowerGear and XPE. ① For ATAPP, 15% designs are split out for testing purpose, these designs across all the FPGAs are not used during the training. When it comes to the actual training, we split all the data samples from one FPGA out, and train the model with the rest of the data samples. Our model is tested with the separate designs on the separate FPGA to ensure that both designs and FPGA are unseen to our model. This **level-one-out** strategy is iterated and applied to all 8 FPGAs. Specifically, all the designs on one FPGA are reserved and not used for training, 85% designs on the rest FPGAs are used for training. 15% designs on the reserved FPGA are used for testing. Therefore, 8 models are generated independently and each model is tested with unseen designs on an unseen FPGA. ② For PowerGear, same 15% designs are split out in advance for testing. The model is trained with all the rest designs on FPGA8 and tested on all other FPGAs with the separate design. For the result on FPGA8, the model is trained with the designs on FPGA1. ③ For XPE results, we use the HLS estimated resource and default toggling rate 12.5% to conduct the power estimation across all the FPGAs. We do not use any post-HLS reports to make sure the power estimated by three methods are from before the HLS stage. In this way a fair comparison is guaranteed.

We have several key observations from the table and figure. First of all, our ATAPP significantly outperforms original prior works for all the power estimations on 8 FPGAs with much lower RRSE and MAPE and higher correlation R. Secondly, the correlation of all three methods are larger than 0.5. It can be seen from Fig. 7 that all three methods follow the same trend as the ground-truth power and it implies that all the three methods can correctly capture the design complexity. Meanwhile, ATAPP is more aligned with the ground-truth line with higher R value. It is expected that most of the green dots (XPE estimation) are beyond the ground-truth line since XPE tends to estimate worst-case power. Thirdly, XPE performs the worst among the three methods due to the poor estimation of switching activity. PowerGear can infer the switching activity from high-level code simulation which leads to better estimation than XPE, however, ATAPP performs better with the encoding of the FPGA architecture. Fourthly, ATAPP performance is quite stable across the experimental FPGAs: MAPEs are varying within [12.74%, 13.49%] (smaller is better) and RRSEs are within [0.20, 0.24] (smaller is better). The correlation metrics of ATAPP are around 0.90 (larger is better). Therefore, ATAPP has good generalizability.

Further experiments are conducted to study the effects of the positional encoding on our model performance. We remove the ❺ **Positional Encoding** step in subsection IV-A and use the compressed embeddings directly to train and test the model. Results are shown

TABLE VI
ACCURACY COMPARISON OF DIFFERENT METHODS
ATAPP: PREDICTION WITH DESIGN AND ARCH FEATURES (POS ENC=POSITIONAL ENCODING); **LEAVE-ONE-OUT** FOR TRAINING AND TESTING;
POWERGEAR: TRAIN WITH THE DESIGNS ON FPGA8 AND TEST WITH THE DESIGNS ON FPGA1-FPGA7;TRAIN ON FPGA1 AND TEST ON FPGA8;
XPE: ESTIMATES WITH DESIGN CHARACTERISTICS ON EACH FPGA SHEET

Settings for ATAPP		ATAPP w/ Pos Enc			ATAPP w/o Pos Enc			PowerGear			Vivado+XPE		
Test	Training Set	MAPE	RRSE	R	MAPE	RRSE	R	MAPE	RRSE	R	MAPE	RRSE	R
FPGA1	FPGA 2-8	13.27%	0.22	0.89	27.39%	0.48	0.81	44.56%	0.74	0.76	471.93%	9.73	0.66
FPGA2	FPGA 1, 3-8	13.35%	0.21	0.90	28.48%	0.47	0.80	54.00%	0.72	0.61	385.64%	5.34	0.58
FPGA3	FPGA 1-2, 4-8	13.18%	0.23	0.89	26.86%	0.46	0.79	48.43%	0.68	0.71	395.37%	6.74	0.59
FPGA4	FPGA 1-3, 5-8	12.75%	0.24	0.90	29.39%	0.48	0.82	40.22%	0.62	0.74	390.07%	7.58	0.62
FPGA5	FPGA 1-4, 6-8	13.12%	0.20	0.88	26.42%	0.45	0.81	41.83%	0.61	0.74	405.17%	6.86	0.60
FPGA6	FPGA 1-5, 7-8	12.74%	0.23	0.90	26.54%	0.47	0.81	43.09%	0.67	0.74	419.32%	6.96	0.64
FPGA7	FPGA 1-6, 7	12.78%	0.20	0.91	28.76%	0.48	0.79	42.59%	0.60	0.74	387.96%	6.99	0.65
FPGA8	FPGA 1-7	13.49%	0.23	0.89	27.38%	0.46	0.79	42.28%	0.64	0.75	387.79%	5.85	0.52
Average		13.09%	0.22	0.89	27.34%	0.49	0.80	44.63%	0.66	0.72	405.41%	7.01	0.61

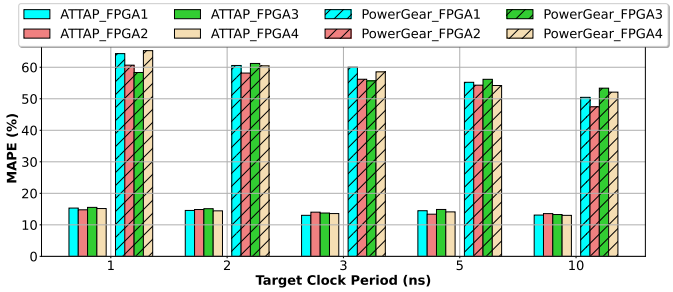

Fig. 8. Both **ATAPP** and **PowerGear** are tested on designs with different target clock periods on different FPGAs. The training settings, **leave-one-out** strategy, are the same as shown in Table VI.

in the columns ATAPP w/o PE of Table VI. Compared to PowerGear and XPE, it shows that ATAPP is able to provide better performance with lower MAPE ranging within [26.42%, 29.39%], lower RRSE [0.45, 0.49] and higher R [0.79, 0.81] due to the introduction of FPGA architecture representation. However, ATAPP w/ PE performs even better due to the feature embeddings with additional resource positions over the FPGA fabric.

D. Robustness study on clock period

Designs generated by HLS are greatly affected by the user-defined target clock period. Even with the same pragma and directive settings, the HLS tool can still generate different designs depending on the constraints, and one of the important constraints is the target clock period. In order to achieve the target clock period, HLS tools perform different operator scheduling and therefore generated designs are different. We conducted further experiments to test the model sensitivity to the designs synthesized under different target clock period settings and results are shown in Fig. 8.

We used ATAPP which is trained with **leave-one-out** strategy to guarantee one FPGA is unseen to the model (all the designs on one FPGA are reserved and not used for training, 85% designs on the rest FPGA are used for training), 4 models (FPGA1-4) are generated independently. Every model is tested with 100 designs selected from the rest 15% designs in its reserved FPGA. Since most of the designs in the training are synthesized with the same target clock period (10ns), we explicitly tune the 100 designs with different target clock periods, we use the same pragmas/settings but different target clock periods (1ns, 2ns, 3ns, 5ns, 10ns) to generate 500 designs (100 designs for each clock period) per FPGA. The 100 designs on each FPGA are used for testing the model sensitivity and results are shown in the Fig. 8. It is observed that ATAPP performs the best (∼13%) when making prediction for the designs with 10ns target clock period due to the training designs are under the same configuration, but when testing for the designs on the other target clock period, ATAPP can still maintain similar performance (13% - 15%) since the design representation used in ATAPP is generated at post-HLS stage and it covers sufficient information of the generated designs with different clock period.

In comparison, we also tested PowerGear with these designs and results are shown in the bars of Fig. 8, the model is trained with 85% designs on FPGA8. PowerGear is not designed for predict power for unseen FPGAs, therefore, we can observe an obvious accuracy degradation compared to ATAPP. There is a similar observation that the model performs the best on the designs synthesized with 10ns target clock period but maintain close MAPE for other designs because PowerGear takes achieved clock period from HLS reports as one of the features in the prediction.

VI. CONCLUSION

In this paper, we present ATAPP, an architecture and technology aware power predictor for unseen FPGA architecture and designs. The proposed method combines both FPGA architectural representation and design representation to predict average dynamic power. ATAPP extracts FPGA architectural features at the tile level to form embeddings and further augments these embeddings with a positional encoding mechanism to generate a better representation. The design representation is a graph generated from the IR data flow with FSMD model and operator switching activities at IR level. Our experiments show that ATAPP provides an accurate estimate with 13.9% error in unseen designs with unseen FPGA architectures. We also perform robust studies on clock period to ATAPP and results show that our model can maintain similar error with designs synthesized using different target clock periods.

VII. ACKNOWLEDGEMENT

We thank all anonymous reviewers for their detailed comments on the paper and Nanditha Rao for her help on RapidWright exploration. This work was partially supported by the National Science Foundation grants 2326894, 2425655, and 2417658. Any opinions, findings, conclusions or recommendations are those of the authors and do not necessarily reflect the views of these funding agencies. The authors also acknowledge the computing resources provided by the Texas Advanced Computing Center (TACC).

REFERENCES

[1] 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.

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

[4] "AMD Power Estimator (XPE)." [Online]. Available: https://www.amd.com/en/products/adaptive-socs-and-fpgas/technologies/power-efficiency/power-estimator.html

[5] K. E. Murray, O. Petelin, S. Zhong, J. M. Wang, M. ElDafrawy, J.-P. Legault, E. Sha, A. G. Graham, J. Wu, M. J. P. Walker, H. Zeng, P. Patros, J. Luu, K. B. Kent, and V. Betz, "VTR 8: High Performance CAD and Customizable FPGA Architecture Modelling," *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, 2020.

[6] 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.

[7] 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.

[8] 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.

[9] 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.

[10] 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.

[11] 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.

[12] 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.

[13] 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.

[14] 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.

[15] 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.

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

[17] 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.

[18] 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.

[19] 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.

[20] 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.

[21] 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.

[22] 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.

[23] 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.

[24] 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.

[25] 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.

[26] 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.

[27] 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.

[28] 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.

[29] 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.

[30] 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.

[31] 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.

[32] 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.

[33] 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.

[34] 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.

[35] 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.

[36] 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.

[37] 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.

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

[39] 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.

[40] 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.

[41] 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.
[42] C. Lavin and A. Kaviani, "RapidWright: Enabling Custom Crafted Implementations for FPGAs," in *International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, 2018.
[43] 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.
[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] 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.
[47] 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.
[48] L.-N. Pouchet, "Polybench: The polyhedral benchmark suite," 2012. [Online]. Available: http://web.cs.ucla.edu/ pouchet/software/polybench/
[49] S. Abi-Karam, R. Sarkar, A. Seigler, S. Lowe, Z. Wei, H. Chen, N. Rao, L. John, A. Arora, and C. Hao, "HLSFactory: A Framework Empowering High-Level Synthesis Datasets for Machine Learning and Beyond," in *ACM/IEEE International Symposium on Machine Learning for CAD (MLCAD)*, 2024.