Active Learning-based Practical Power Estimation Considering Multi-Cycle Paths

Shao-Min Liu[1], Shao-Yun Fang[1], Hsiang-Wen Chang[2], Ming-Chao Lee[2], and Peter Wei[2]
[1]National Taiwan University of Science and Technology, Taipei, Taiwan
[2]Synopsys Inc., Taiwan

***Abstract*—In order to meet the design requirements of low-power products, how to provide an accurate power estimation in early stages of the design flow has become more and more important. A previous research method uses the toggle rates of registers to perform design-dependent RTL power estimation based on machine learning (ML) techniques. However, the runtime speedup claimed by the previous work ignores the huge runtime obtaining the labels for training data, making the ML-based approach insufficiently efficient for general designs. In this paper, we adopt active learning to query the labels of most representative training data and propose a new feature representation approach to enhance the model accuracy given that the training data are deficient. A recurrent neural network (RNN)-based autoencoder is also adopted, which makes the proposed model able to handle the designs with multi-cycle paths. Experimental results show that compared to the existing work, the proposed training flow can greatly improve the power estimation accuracy with much fewer training data.**

I. Introduction

Currently, power analysis in later design stages mainly uses various netlist characteristics and physical layout information that have been designed to calculate the power consumption of a circuit. However, as the demand of low-power products and the design complexity dramatically increase, gate-level power analysis becomes so time-consuming that only thousands out of millions or cycles can be simulated in reasonable runtime. Therefore, a fast yet sufficiently accurate register transfer level (RTL) power analysis approach is required to first obtain the power profile over millions of cycles, and then the identified critical cycles with higher estimated power are further simulated in gate-level power analysis.

Previous research mainly focuses on a key element that greatly determines dynamic power consumption of a circuit, that is, switching activities. The switching activity of a circuit node is the number of toggles per clock cycle on the signal, averaged across many clock cycles. The switching transitions of the registers in a design are usually regarded as the main index of circuit power consumption. The power estimation problem is usually divided into two stages: the training stage and the estimation stage. In the training phase, the actual power consumption is calculated by observing the activity of a certain part of a circuit and simulating a certain number of cycles. The goal is to find the relationship between the switching activities of registers and the corresponding power trace. Therefore, the coefficient of each register can be trained to represent the contribution of this register to the overall power consumption. [1] and [2] respectively use linear regression and regression tree methods to characterize the power consumption of a circuit module (IP) or a small entire circuit by considering all registers. [3] uses a feature selection technique based on singular value decomposition (SVD) to characterize the power consumption of an entire circuit after selecting the key registers.

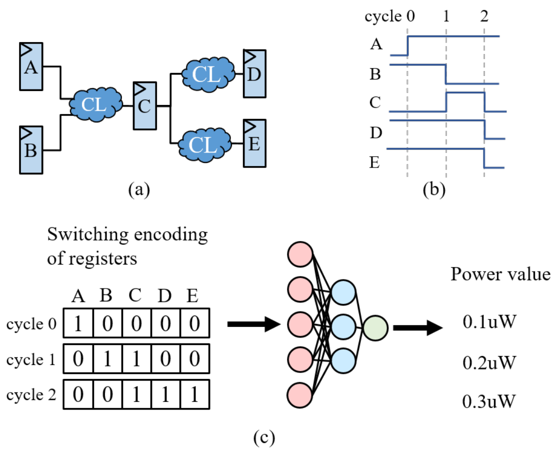

Fig. 1. The key idea of the state-of-the-art work [4]. (a) A sample netlist. (b) The switching waveforms of the five registers. (c) Switching encoding and the ML-based power estimation approach.

The state-of-the-art studies [4] and [5] use machine learning (ML)-based methods to further enhance the efficiency and accuracy of power estimation. [4] proposes a fast ML-based power estimator called PRIMAL, which adopts several ML models such as multi-layer perceptron (MLP) and convolutional neural network (CNN) for average power and cycle-by-cycle power estimation at RTL or behavior level. It is reported in [4] that cycle-by-cycle power estimation suffers from larger prediction errors than average power estimation. [5] develops a GPU-accelerated graph neural network (GNN) model to perform average power estimation for gate-level netlists. As mentioned earlier, the major objective of our work is to identify critical cycles resulting in great power for RTL. Therefore, PRIMAL is the target state-of-the-art work that will be detailed in the following and compared in the experiments. Fig. 1 illustrates the idea proposed in PRIMAL. For a given RTL netlist in Fig. 1(a), the switching waveform of each register (as well as each I/O signal) is also given, as shown in Fig. 1(b), where each edge corresponds to a clock rising edge. For cycle-by-cycle power estimation, the switching states of all registers and I/O signals at a single clock rising edge are transferred into a switching encoding, as illustrated in Fig. 1(c), where a switching or a non-switching event is represented by a binary bit or few binary bits. By regarding the switching encoding as the input of a machine learning model, an estimated power value can be derived at the output.

As mentioned in [4], the application of PRIMAL is limited to reusable intellectual properties (IPs). The major reason is that PRIMAL trains the estimation model with the design testbench of an IP, and then the model is applied when the IP is reused (or embedded in another design) with various workloads. The great runtime speedup

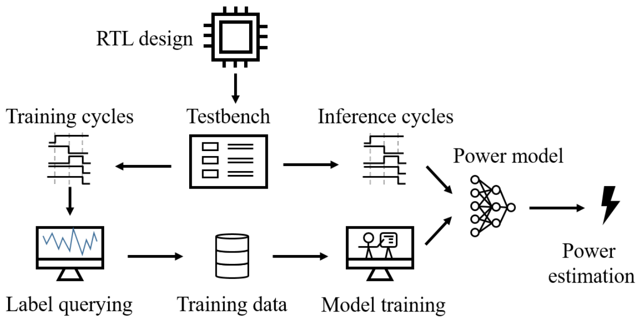

Fig. 2. The application of ML-based power estimation for general designs.

(50X on average) is achieved when the IP is reused. However, for a general design whose power analysis is only performed on the design testbench, the speedup may not be so significant if a large amount of training data is required. Fig. 2 illustrates a practical power estimation scenario with ML methods. For example, if 80% of clock cycles in the testbench are used for training, the labels (ground-truth power values) of the 80% data are still derived by running the power analysis with an EDA tool. Even if the rest 20% data can be efficiently inferenced, the overall runtime speedup must be less than 1.25X. Obviously, using 80% of the data for model training is very unfavorable for fast power estimation of general designs.

Recently, the concept of active learning has been widely adopted in ML-related studies especially in the situation that obtaining data labels is either expensive or time-consuming. In active learning, a relatively small set of unlabeled data with the largest amount of hidden information is selected. Performing labeling for the selected data and training an ML model based on the selected data, the ML model with comparable accuracy to the model trained by using all data with labels is expected. In this paper, we propose an active learning approach that can use less training data to derive an RTL power estimation model with sufficient accuracy, and thus more speedup can be achieved in power analysis for general designs. Our main contributions are as follows:

- A new switching encoding method is proposed to improve prediction accuracy. In addition, the features for glitch power consumption can also be integrated if gate-level circuit information is available.
- A recurrent neural network (RNN)-based auto-encoder is adopted in the prediction model to handle the designs with multiple-cycle (multi-cycle) paths.
- Active learning techniques are proposed to select representative data to train the power estimation model and provide sufficient model performance.
- Experimental results show that the proposed switching encoding and the data selection mechanisms make our model outperform a state-of-the-art work, and the adopted RNN-based auto-encoder can further enhance estimation accuracy for the designs with multi-cycle paths.

The rest of this paper is organized as follows: Section II introduces the proposed methodologies. Section III presents the experimental results. Finally, a short conclusion is given in Section IV.

II. Proposed Enhancement Method

In this section, the proposed model architecture, the new feature representation, the multi-cycle path handling, and the active learning-based training flow are separately detailed in the following subsections.

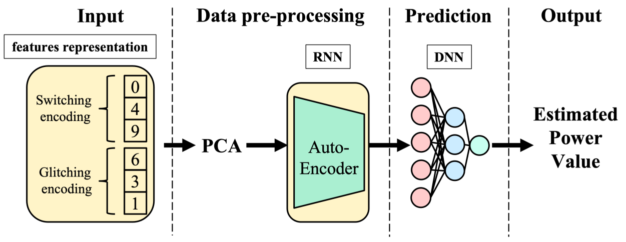

Fig. 3. The proposed model architecture.

A. Model Architecture

The proposed model architecture is shown in Fig. 3. A new switching encoding is used to differentiate different types of switching transitions. In addition, if gate-level circuit information is available, a glitching encoding is also constructed and cascaded after the switching encoding. Due to the huge dimension of the input feature vector, principal components analysis (PCA) is first applied for dimension reduction and to avoid overfitting. The reduced dimension of the feature vector is 512×1, which is fixed for any input circuit. For the design of multi-cycle paths, an RNN-based auto-encoder is adopted to reduce the dimension of each feature vector from $512 \times (k + 1)$ of $k + 1$ consecutive cycles to a feature vector of 512×1. After that, each reduced feature vector is fed into a deep neural network (DNN) composed of three MLPs. Finally, an estimated power value can be derived at the model output.

B. Switching Feature Representation

Due to the high correlation with dynamic power, the switching activities of registers are the key features and have been commonly used in power estimation models in previous work. The switching encoding adopted in the MLP model of [4] only records whether a register toggles in a clock cycle or not. For the CNN model proposed in [4], the switching encoding only categorizes each register as non-switching, rising transition, or falling transition in each clock cycle. In order to better learn the relationship between switching activities and the power consumption, we explicitly define 16 types of switching transitions. Each type of transitions consists of an original state (FROM state) and a switched state (TO state), as shown in Fig. 4. Each of the transition types including an unknown FROM state or an unknown TO state is encoded as 0, because the power consumption of a switching transition of these types is not considered and counted in the adopted power analysis tool. In addition, each of the other transition types is encoded as one of the integer between 1 and 16. Empirically, how to assign the 16 integers to the 16 transition types does not significantly affect the model performance, and thus the 16 integers are only used for differentiate the differences among the 16 transition types.

C. Glitching Feature Representation

It can be observed from some industrial benchmarks that glitching power may account for a large portion of total power. However, signal glitches cannot be derived before a gate-level netlist is synthesized. Therefore, once the gate-level circuit information is available, the proposed model can also be applied to cycle-by-cycle gate-level power estimation. Since glitch power is only consumed by combinational cells and may cause multiple and additional signal toggles in a single cycle, another feature representation method is required to characterize glitches. Recording the number of toggles for each combinational cell is not applicable since the dimension of feature vectors will explosively increase. Therefore, for each combinational cell, the corresponding cell type in the standard cell library is

		TO state				
		U	X	0	1	Z
FROM state	U	0	0	0	0	0
	X	0	4	5	8	12
	0	0	7	1	10	14
	1	0	6	9	2	16
	Z	0	11	13	15	3

U: unknown X: floating Z: high impedance

Fig. 4. The table of 16 transition types and their encoding used in our model, each of which consists a FROM state and a TO state.

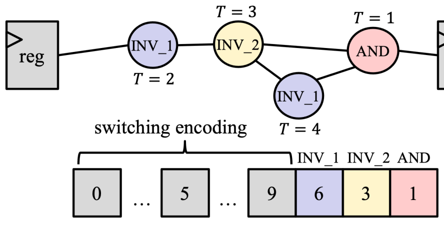

Fig. 5. The proposed glitching feature encoding added after the switching encoding.

identified, and the number of additional signal toggles of each cell is accumulated to the corresponding cell type for a single clock cycle. Fig. 5 illustrates an example, where T indicates the number of additional signal toggles at the output of each combinational cell. Since there are two cells of the same type INV_1, the number of toggles of INV_1 is computed by summing up the toggles of the two cells. The feature representation vector with glitch consideration can be constructed by cascading the glitching encoding after the switching encoding.

D. Multi-cycle Path Handling

During the development of our power estimation model, we found that two very similar (the similarity ¿ 99%) feature vectors can result in very different power values (power difference ¿ 30%), which can cause great errors in the cycle-by-cycle power estimation process. Our investigation on this problem shows that it may be due to the multi-cycle paths in advanced industrial designs, where the delays of some signal paths specified by designers are allowed to be more than one clock cycle. The existence of multi-cycle paths in a design may cause the power value of a target cycle significantly affected by the switching activities of the previous cycles. Since the switching activities of successive clock cycles are not independent, it is desirable to simultaneously consider multiple feature vectors for a design with multi-cycle paths. However, directly cascading multiple feature vectors greatly increases the input dimension of the DNN, which not only causes difficulties in training but may cause overfitting. Thus, maintaining the dimension of feature vectors while considering successive clock cycles is a preferred option.

To achieve this goal, we adopt an RNN-based auto-encoder, which is originated from the RNN encoder–decoder proposed in [11]. The RNN encoder–decoder is a neural network composed of a "many-to-one" RNN encoder and a "one-to-many" RNN decoder, as shown in Fig. 6. The encoder compresses the input vector X into an intermediate vector Z, and the decoder restores Z to a reconstructed

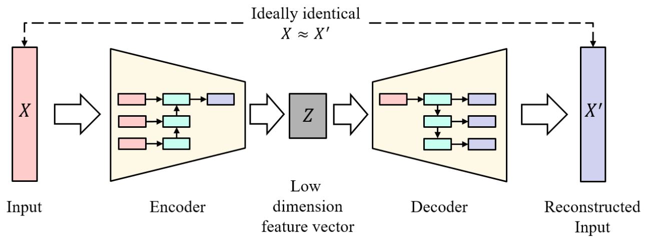

Fig. 6. An encoder–decoder architecture composed of a "many-to-one" RNN encoder and a "one-to-many" RNN decoder.

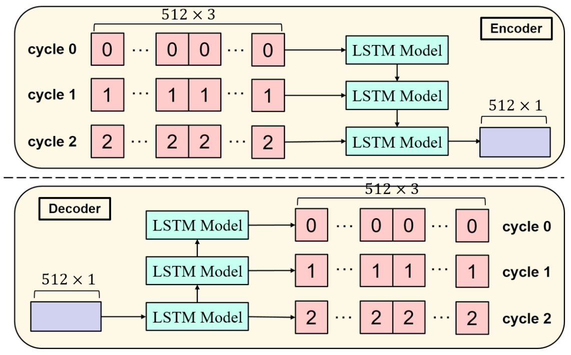

Fig. 7. The training phase of the implemented RNN-based auto-encoder with $k = 2$.

vector X'. By minimizing the error between X and X', Z can be regarded as a reduced version of X with a lower dimension.

Fig. 7 shows the RNN encoder–decoder used in our network. Long Short-Term Memory (LSTM) is used to alleviate the gradient explosion/vanish problem. Suppose k additional clock cycles need to be considered for each cycle. After PCA, the feature vectors of $k + 1$ cycles are fed into the encoder. Fig. 7 shows an example with $k = 2$, where the red vectors are the feature vectors of the three cycles, the blue vectors, are RNN hidden units, and the purple vector is the dimension-reduced feature vector. The encoder compresses the features considering three cycles into a vector of the same size as the feature vector of a single cycle. After the training process of the RNN encoder–decoder architecture, the decoder network is removed, and the encoder network serves as a dimension reducer for the feature vectors considering multiple cycles.

E. Active Learning-based Training Flow

To reduce the required training data and achieve sufficient estimation accuracy in our power model, active learning that queries the most representative data is applied. [6] proposes to transform an active learning problem into a core-set selection problem, which can outperform many active learning heuristics in literature. As shown in Fig. 8, the data are distributed on an established plane according to their similarities. The blue points are a set of selected data to be labeled, and the red points are the remaining unlabeled data. Regarding the selected data as the centers, a set of circles with the same radius r_c is drawn such that all the remaining unlabeled data points are covered by the circles. The results in [6] show that the blue selected point can be used as a representative point for the remaining red points in each circle. In addition, if the selection of a point to be labeled does not reduce r_c, it will not improve the accuracy of model training. The core-set problem asks to find a set of selected data that can minimize r_c within the selection budget.

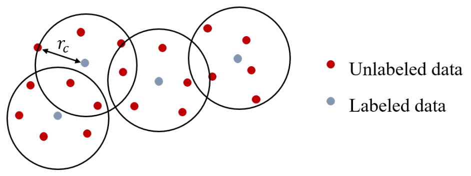

Fig. 8. Active learning by solving a core-set problem.

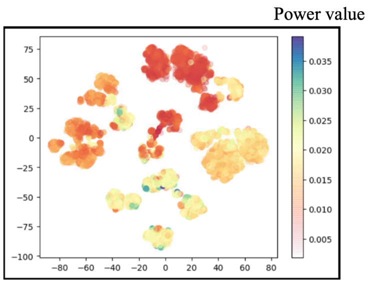

Fig. 9. 2D visualization of feature vectors with t-SNE.

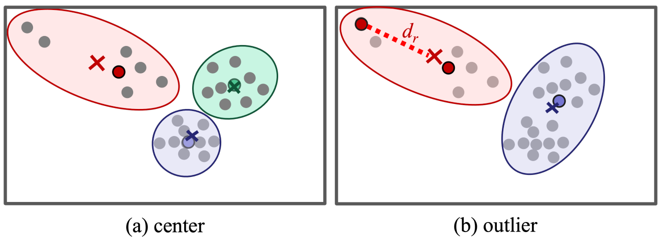

Fig. 10. Two data selection mechanisms: (a) CENTER and (b) OUTLIER.

Fig. 9 gives the 2D visualization result of the feature vectors extracted in an industrial design by performing t-distributed stochastic neighbor embedding (t-SNE). It can be found in Fig. 9 that the new feature representation approach makes feature vectors with high similarity have similar power values. As a result, it is applicable to solve the core-set problem as our active learning oracle as well.

The core-set problem was solved with a k-Center-Greedy algorithm in combination with a mixed integer program (MIP) formulation in [6]. However, solving our problem with MIP is not practical due to its prohibitively long runtime. As a result, we alternatively use the well-known k-means algorithm for data clustering and use the k centers derived from the k-Center-Greedy algorithm as the initial cluster centers of the k-means algorithm. The k-Center-Greedy algorithm can speedup the convergence of the k-means algorithm. Two data selection mechanisms are proposed as follows:

-1- ***CENTER***: this selection mechanism selects the data point closest to the center of each cluster as the representative data point. Therefore, the number of clusters is set as the number of data selected for training. An example illustration is given in Fig. 10(a).

-2- ***OUTLIER***: empirically, in addition to select the points closest to the cluster centers, selecting some outlier points can result in better accuracy in some cases. As shown in Fig. 10(b), as the number of clusters decreases, some data points which are farthest away from cluster centers are also selected.

The entire training process is summarized in Fig. 11. Input is an original VCD file with a long switching waveform to be simulated for power analysis. Feature extraction with the new representation method and PCA are performed for each clock cycle. For a design with multi-cycle paths, the RNN auto-encoder is adopted to enable multi-cycle consideration while maintaining the reduced size of feature vectors. Then, data selection is applied using one of the above mechanisms. The selected data are simulated with PrimePower in PTPX [14] to derive their ground truth power values. Finally, the DNN model is trained with the labeled data, and the power values of the cycles that are not selected in the active learning step can be estimated by using the trained model.

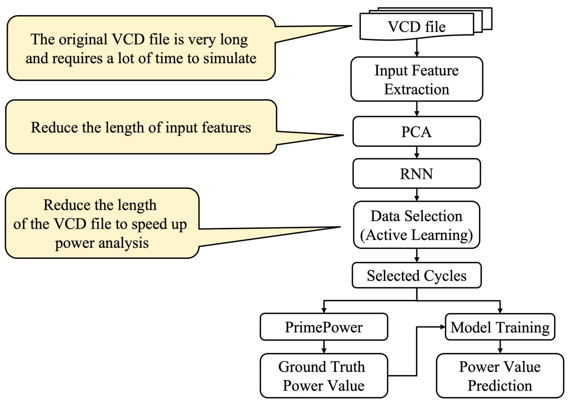
The original VCD file is very long and requires a lot of time to simulate
VCD file
Input Feature Extraction
Reduce the length of input features
PCA
RNN
Reduce the length of the VCD file to speed up power analysis
Data Selection (Active Learning)
Selected Cycles
PrimePower
Model Training
Ground Truth Power Value
Power Value Prediction

Fig. 11. The training flow proposed in this paper.

III. Experimental Results

The process of parsing a VCD file and constructing the feature representation vectors are implemented with the C++ programming language, and the machine learning-based cycle-by-cycle power estimation framework is implemented with Python 3.7. The K-means algorithm is implemented with *scikit-learn* [13], and the other models are implemented with *Keras* [12]. All experiments were conducted on a Linux machine with Intel Xeon Gold 6248 CPU @ 2.5 GHz and 354 GB memory. The power waveform is obtained from the PrimePower of Synopsys PTPX [14] for data labeling. Table I lists three adopted industrial benchmark circuits, where the numbers of registers and I/O signals (defined in the VCD files), the numbers of circuit gate counts, the circuit clock periods (ps), the numbers of waveform cycles, and the runtimes for full power analysis in PTPX are separately given in Rows 2–6. Among the three cases, *Design_A* is a relatively small design, and *Design_B* and *Design_C* are two large-scale designs, each of which contains more than 200k registers and I/O signals. The VCDs have accounted for workload variety and thus the power values are greatly diverse in both training and testing data. More detailed information regarding the benchmark circuits is not allowed to be revealed due to the IP issue.

We compare the performance of the proposed ML-based techniques for power estimation with the previous work [4]. Two models,

TABLE I
Statistics of the Benchmark Circuits.

Benchmark	*Design_A*	*Design_B*	*Design_C*
#Reg.+I/O Signals	410	208,850	216,284
#Cell Gates	2,391	603,114	326,164
Clock Period (ps)	4,000	952	10,000
#Total cycles	14,999	11,118	9,737
PTPX Run Time (s)	1,232	7,986	17,169

MLP and CNN, are majorly used and compared in [4]. We implement the two models (the implemented MLP is the same as the DNN architecture in our model for fair comparison) and report the better results in the following tables. Note that we do not compare our work with [5] since this existing work focuses on average power prediction for a window (a period of clock cycles) of interest, while our aim is to perform cycle-by-cycle peak power estimation. The feature extraction approach as well as the model architecture proposed in [5] are not applicable in cycle-by-cycle RTL power estimation.

A. *Effectiveness of Active Learning and the Proposed Switching Encoding*

First, we evaluate the effectiveness of the proposed feature representation approach and the active learning method. The accuracy of power estimation is measured by normalized root mean square error (NRMSE). The results of *Design_A*, *Design_B*, and *Design_C* are respectively reported in Tables II, III, and IV, where the results derived from different ratios of training data (TD) are shown in separate columns, and "Error" and "Ratio" respectively denote the NRMSE and the error ratio compared to the baseline work. The results of "10% TD" are not reported for *Design_B* since no approach can achieve reasonable estimation accuracy. Four methods are first compared in the three tables: "[4]" uses the binary switching encoding in the MLP model or the three-state switching encoding in the CNN model proposed in [4], and the training data are randomly selected (baseline); "[4]+CENTER" uses the same switching encoding as that of "[4]" and the "CENTER" data selection scheme proposed in our work; "Ours_CENTER" uses the 16-transition-type switching encoding, the model architecture excluding the RNN-based autoencoder, and the "CENTER" data selection scheme proposed in our work; "Ours_OUTLIER" is the same as "Ours_CENTER", while the "OUTLIER" data selection scheme is adopted. Due to the different selection mechanisms, the testing data of different methods are not the same. For fair comparison, the NRMSEs listed in the tables are the NRMSEs of all cycles (i.e. training set + testing set). Note that the NRMSE improvements derived from our approach are more significant if only testing set is counted (about 2X improvements compared to those made in Tables II–IV). Observing from the tables, for the relatively small design *Design_A*, the proposed 16-transition-type switching encoding and active learning-based selection mechanism are helpful for model training when a small ratio of training data (10% or 20%) is selected, where 32% and 9% accuracy improvement can be achieved by one of the two data selection methods. For the other two large-scale designs *Design_B* and *Design_C*, the accuracy improvements are more significant. For example, the accuracy improvements can be up to 76% and 47% respectively for *Design_B* and *Design_C* when 40% data are selected for training, and 85% and 63% improvements can respectively be achieved when 80% data are selected.

The great accuracy improvements achieved in large-scale designs can, for example, be understood by the scatter charts of *Design_B* with "80% TD" shown in Fig. 12, where the x-axis is the golden power values and the y-axis is the estimated power values. When the data are more scattered on the 45° line, the estimated values are more accurate. The highlighted red points are the real data with the top 30 power values. The correct estimations of these data with higher power values can contribute to the power analysis and signoff in the later design stages. Figs. 12(a) and (b) respectively show the scatter charts of training data for [4] and our approach, and Figs. 12(c) and (d) respectively show those of testing data. It can be found that although [4] can achieve fine estimation accuracy on training data, it suffers from large NRMSE on testing data. In contrast, our approach performs well on both training data and testing data.

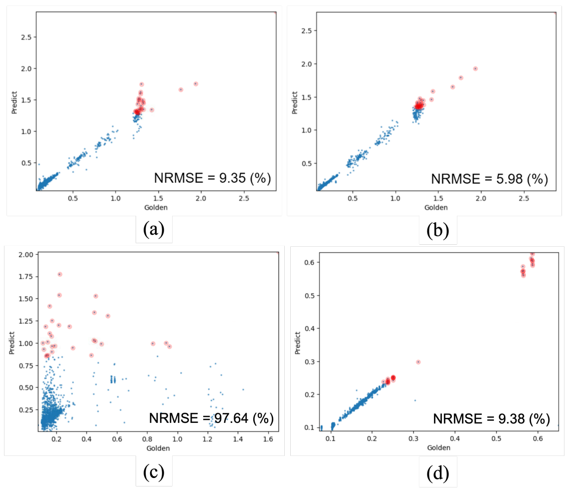

Fig. 12. The scatter charts of *Design_B* with 80% TD. (a) Training data of [4]. (b) Training data of our approach. (c) Testing data of [4]. (d) Testing data of our approach.

B. *Effectiveness of the RNN-based Auto-Encoder*

We show the effectiveness of adopting the proposed RNN-based auto-encoder for multi-cycle path handling. Since only *Design_B* and *Design_C* contain multi-cycle paths, the results are only shown in Tables III and IV. In the two tables, "Ours_CENTER +RNN_4" indicates $k = 4$ in the RNN-based auto-encoder, which means that the proposed model considers the switching features of five consecutive clock cycles. The "CENTER" selection scheme is used in this experiment. Similarly, "Ours_CENTER +RNN_8" indicates $k = 8$ in the RNN-based auto-encoder. The results show that the adoption of the RNN-based encoder can further enhance estimation accuracy when fewer proportions of data are selected for training ($\leq$ 40% TD). Compared to the method "Ours_CENTER", for example, an additional 11% accuracy improvement is obtained in *Design_B* with "40% TD", and an additional 44% accuracy improvement is obtained in *Design_C* with "20% TD". It can also be found that when the proportion of training data increases, using $k = 8$ in the auto-encoder decreases estimation accuracy, which may be due to the training error of the auto-encoder itself.

C. *Effectiveness of the Glitching Encoding*

This section further shows the effectiveness of the proposed glitching encoding. Note that this encoding can only be applied when gate-level circuit information is available and is also applicable in cycle-by-cycle power estimation. The results are also reported for the two large-scale designs in Tables III and IV, where the two sets of data respectively denoted by "glitch_I" and "glitch_O" indicate that each glitch belongs to the input of the cells behind or belongs to the output of the cell ahead. Both the two methods count the number of rising transitions in each glitch and use the "CENTER" data selection method. The results in Table III do not show great accuracy improvement when the glitching encoding is adopted, because the glitching power in *Design_B* does not contribute much in the total power. On the other hand, the results in Table IV show that the glitching encoding can further enhance the estimation accuracy when training data are sufficient ($\geq$ 40% TD). In addition, counting the glitches at cell inputs usually leads to better performance than counting them at outputs.

TABLE II
The NRMSEs of different approaches with different ratios of training data for Design_A.

Methods	10% TD		20% TD		40% TD		60% TD		80% TD	
	error (%)	Ratio	Error (%)	Ratio	Error (%)	Ratio	Error (%)	Ratio	Error (%)	Ratio
[4]	10.5	1.00	6.2	1.00	5.4	1.00	5.2	1.00	5.2	1.00
[4]+CENTER	8.3	0.79	7.3	1.17	6.6	1.22	6.7	1.29	6.3	1.21
Ours_CENTER	8.6	0.82	**5.7**	**0.91**	5.5	1.01	**5.0**	**0.97**	4.9	0.95
Ours_OUTLIER	**7.1**	**0.68**	5.8	0.94	**5.1**	**0.95**	5.2	1.00	**4.8**	**0.92**

TABLE III
The NRMSEs of different approaches with different ratios of training data for Design_B.

Methods	20% TD		40% TD		60% TD		80% TD	
	Error (%)	Ratio	Error (%)	Ratio	Error (%)	Ratio	Error (%)	Ratio
[4]	133.4	1.00	83.1	1.00	56.8	1.00	45.0	1.00
[4]+CENTER	72.0	0.54	35.9	0.43	18.8	0.33	15.0	0.33
Ours_CENTER	59.6	0.45	28.7	0.35	**9.8**	**0.17**	7.8	0.17
Ours_OUTLIER	65.5	0.49	**20.0**	**0.24**	10.6	0.19	**6.7**	**0.15**
Ours_CENTER+RNN_4	**55.0**	**0.41**	20.1	0.24	9.9	0.17	10.2	0.23
Ours_CENTER+RNN_8	58.4	0.44	24.8	0.30	17.8	0.31	16.9	0.38
Ours_CENTER+glitch_I	55.7	0.42	19.2	0.23	8.4	0.15	7.3	0.16
Ours_CENTER+glitch_O	55.9	0.42	21.9	0.26	8.4	0.15	7.6	0.17

TABLE IV
The NRMSEs of different approaches with different ratios of training data for Design_C.

Methods	10% TD		20% TD		40% TD		60% TD		80% TD	
	Error (%)	Ratio	Error (%)	Ratio	Error (%)	Ratio	Error (%)	Ratio	Error (%)	Ratio
[4]	17.7	1.00	12.0	1.00	9.3	1.00	8.4	1.00	7.8	1.00
[4]+CENTER	17.6	0.99	10.3	0.86	7.6	0.81	7.2	0.86	7.2	0.92
Ours_CENTER	16.9	0.95	12.1	1.01	4.9	0.53	**3.4**	**0.40**	**2.9**	**0.37**
Ours_OUTLIER	18.0	1.01	10.5	0.88	5.5	0.59	4.5	0.53	3.7	0.48
Ours_CENTER+RNN_4	10.9	0.61	7.1	0.59	**4.8**	**0.51**	3.6	0.42	**2.9**	**0.37**
Ours_CENTER+RNN_8	**9.0**	**0.51**	**6.9**	**0.57**	4.8	0.52	3.7	0.44	3.2	0.41
Ours_CENTER+glitch_I	12.7	0.72	8.5	0.71	3.1	0.34	1.5	0.18	1.0	0.13
Ours_CENTER+glitch_O	14.6	0.82	8.4	0.70	3.8	0.41	2.1	0.25	1.3	0.16

IV. Conclusion

In this paper, we propose a machine learning-based cycle-by-cycle power estimation model with multi-cycle path consideration. With the proposed active learning mechanisms, the amount of required training data can be reduced with sufficient estimation accuracy, and thereby the time for running a power analysis tool for data labeling can be reduced. The experimental results show that the proposed switching encoding and the data selection mechanisms make our model outperform an existing method, and the adopted RNN-based auto-encoder can further enhance estimation accuracy for the designs with multi-cycle paths when fewer proportions of data are selected for training.

References

[1] Alessandro Bogliolo, Luca Benini, and Giovanni De Micheli, “Regression-based RTL power modeling,” *ACM Transactions on Design Automation of Electronic Systems*, vol. 5, no. 3, pp. 337—372, 2000.

[2] Jason H. Anderson and Farid N. Najm, “Power estimation techniques for FPGAs,” *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 12, no. 10 pp. 1015–1027, 2004.

[3] Jianlei Yang, Liwei Ma, Kang Zhao, Yici Cai and Tin-Fook Ngai, “Early stage real-time SoC power estimation using RTL instrumentation,” in *Proceedings of Asia and South Pacific Design Automation Conference*, 2015.

[4] Yuan Zhou, Haoxing Ren, Yanqing Zhang, Ben Keller, Brucek Khailany, and Zhiru Zhang, “PRIMAL: Power Inference using Machine Learning,” in *Proceedings of Design Automation Conference*, 2019.

[5] Yanqing Zhang, Haoxing Ren, and Brucek Khailany, “GRANNITE: graph neural network inference for transferable power estimation,” in *Proceedings of Design Automation Conference*, 2020.

[6] Ozan Sener and Silvio Savarese, “Active learning for convolutional neural networks: A core-set approach,” in *Proceedings of International Conference on Learning Representations*, 2018.

[7] Burr Settles, ”Active learning literature survey,” in *Computer Sciences Technical Report 1648, University of Wisconsin–Madison*, 2009.

[8] Yasi Wang, Hongxun Yao, Sicheng Zhao, and Ying Zheng, “Dimensionality reduction strategy based on auto-encoder,” in *Proceedings of International Conference on Internet Multimedia Computing and Service*, 2015.

[9] David D. Lewis and William A. Gale, “A sequential algorithm for training text classifiers,” in *Proceedings of the ACM SIGIR Conference on Research and Development in Information Retrieval*, 1994.

[10] Sepp Hochreiter and Jürgen Schmidhuber, “Long short-term memory,” in *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[11] Kyunghyun Cho, Bart van Merrienboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio, “Learning phrase representations using RNN encoder-decoder for statistical machine translation,” in *Proceedings of the Empirical Methods in Natural Language Processing*, 2014.

[12] Keras: The Python Deep Learning library. https://keras.io/, 2018

[13] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, Jake Vanderplas, Alexandre Passos, David Cournapeau, Matthieu Brucher, Matthieu Perrot, and Édouard Duchesnay, “Scikit-Learn: machine learning in python.” *The Journal of Machine Learning Research,*, vol. 12, pp. 2825–2830, 2011.

[14] Synopsys PrimeTime PX. https://www.synopsys.com/support/training/signoff/primetimepx-fcd.html.