Machine Learning Techniques for Pre-CTS Identification of Timing Critical Flip-Flops

Chunkai Fu*, Ben Trombley†, Hua Xiang†, Gi-Joon Nam†, Jiang Hu‡*
*Department of Computer Science and Engineering, Texas A&M University
†IBM T. J. Watson Research Center
‡Department of Electrical and Computer Engineering, Texas A&M University

2023 IEEE Computer Society Annual Symposium on VLSI (ISVLSI) | 979-8-3503-2769-4/23/$31.00 ©2023 IEEE | DOI: 10.1109/ISVLSI59464.2023.10238658

***Abstract*—The timing criticality of flip-flops is a key factor for combinational circuit timing optimization and clock network power reduction, both of which are often performed prior to CTS (Clock Tree Synthesis) and routing. However, timing criticality is often changed by CTS/routing and therefore optimizations according to pre-CTS criticality may deviate from the correct directions. This work investigates machine learning techniques for pre-CTS identification of post-routing timing critical flip-flops. Experimental results show that the ML-based early identification can achieve 99.7% accuracy and 0.98 area under ROC (Receiver Operating Characteristic) curve, and is $62000\times$ to $73000\times$ faster than the estimate with CTS and routing flow on average. Our method is almost $8\times$ faster than a state-of-the-art approach of ML-based timing prediction.**

I. Introduction

Although the pace of Moore's law slows down, chip design complexity continues to grow. For example, the transistor count of Apple's A-series processors increases from 7 billion for A12 in 2018 to 15 billion for A15 in 2021. Such growth generally entails increased design iterations and turn-around time. Reliable yet fast design predictions help steer early optimization steps in the right direction, reduce design iterations and thereby mitigate the challenge of design turn-around time increase.

This work is focused on the early prediction of timing critical flip-flops, which are end-points of timing critical paths. Although flip-flops are not the largest body of circuit elements, they are structurally located at a place of strategic importance – the boundary between combinational logic circuits and clock distribution networks. As such, optimizations involving flip-flops affect both the timing slack of combinational logic paths and clock network power. As chip timing is mostly decided by critical path slack and the clock network is a major chip power consumer, flip-flop optimizations play a critical role in deciding chip timing and power.

A common optimization technique for flip-flops is their placement, which has been employed in industrial timing-driven physical synthesis flows [1]. Flip-flop placement can also be integrated with cloning for further timing improvement [2]. In [3], [4], flip-flop clustering is studied for simultaneous timing optimization and clock network power reduction. Evidently, these techniques need to know which flip-flops are timing-critical *a priori*. While these optimizations are typically performed before routing or even before CTS (Clock Tree Synthesis), the timing criticality of flip-flops is often changed after CTS/routing. Figure 1 depicts pre-CTS slack estimate versus post-routing slack estimate for 1867 flip-flops in a design of 45nm technology, where a dot corresponds to one flip-flop, and the slack values are obtained from a commercial tool. One can see that the correlation between the pre-CTS slack estimate and post-routing slack estimate is quite limited. Also, the pre-CTS slack estimate tends to be very pessimistic due to the use of guard-band. This is consistent with the observation in [5]. As such, flip-flop optimizations guided by the pre-CTS timing estimate may result in over-design and cause unnecessarily more design iterations.

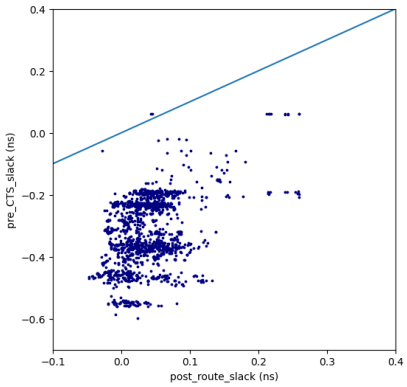

Fig. 1: Pre-CTS vs. post-routing timing slack estimate for flip-flops.

Estimating post-routing timing criticality by going through an entire CTS/routing process is unaffordably expensive for pre-CTS/routing optimizations. Recently, machine learning-based pre-routing timing prediction techniques are developed [5], [6], [7], [8]. The models suggested in [5] are for delay estimates of individual nets and timing slacks are obtained through propagating delays in PERT traversals. As such, its computation runtime is $3\times$ of pre-routing timing analysis by the commercial tool. The machine learning models of [7] rely on post-CTS features. In general, CTS is performed after placement and before routing. Post-CTS timing usually correlates with post-routing timing remarkably better than pre-CTS timing. Hence, predicting post-routing timing using post-CTS features is a relatively easy problem. However, performing CTS entails significant runtime overhead. Both [6] and [8] are regression models using complicated Graph Neural Networks (GNN) or transformer techniques. In [6], it takes more than two seconds for inference on a circuit with

979-8-3503-2769-4/23/$31.00 ©2023 IEEE

234K cells, which is not a large one from an industrial design point of view. Such runtime is not fast enough for frequent use in optimizations at the placement stage. The largest test case (excluding training cases) in [8] has 168K cells and is even smaller.

Although an ML-based timing prediction result can be applied to identify critical flip-flops, there are several important differences:

- It is very difficult, if not impossible, for timing prediction to handle netlist changes caused by circuit optimizations such as buffer insertion and logic reconstruction. For example, the work of [5] predicts the timing of individual nets for a placement solution. However, the nets being predicted may disappear after logic reconstruction. Similarly, the latest work [8] needs to trace graph structures from fixed netlists. By contrast, combinational logic netlist changes do not cause any hindrance to critical flip-flop prediction.
- Timing predictions are generally *regression* inference, and therefore intrinsically more expensive than critical flip-flop predictions, which are *classification* inference. We indeed observed that critical flip-flop prediction is near $8\times$ faster than timing prediction.
- The state-of-the-art timing prediction [8] is based on graph neural network models, whose training time is three orders of magnitude higher than those for critical flip-flop predictions. When ML models need to be retrained due to technology changes, this huge computational cost difference matters.
- Timing prediction and critical flip-flop prediction are prepared for different application scenarios. For instance, timing prediction is useful for timing-driven cell placement that does not involve netlist changes. For design stages where there are still changes in combinational circuits, critical flip-flop prediction facilitates better flip-flop placement and clustering solutions.

This work investigates multiple machine learning techniques for predicting post-routing timing critical flip-flops before CTS. These techniques include logistic regression, neural networks, graph neural network models, and decision tree-based models. The results on several test cases with up to 780K cells show that the best ML technique can achieve an area under ROC (Receiver Operating Characteristic) curve of 0.98 and an accuracy of 99.7%. The contributions of this work are as follows.

- This is the first study on fast early identification of post-routing timing critical flip-flops according to our best knowledge.
- Accurate yet fast identification techniques are found and form a foundation that enables efficient flip-flop optimizations.
- Ablation study is performed to understand the importance of different ML features and model sensitivity to different criticality thresholds.

II. Problem Formulation

We investigate techniques for constructing machine learning models that can predict post-routing timing critical flip-flops prior to CTS. Given a placement solution for a digital circuit design and a slack threshold θ, one such model is expected to identify all and only those flip-flops that have post-routing slacks less than θ. The models perform classification inference, i.e., classifying each flip-flop as either timing critical or not timing critical.

We briefly describe the metrics for evaluating the classification performance of a machine learning model. Inference results can be categorized into TP (True Positive), FP (False Positive), TN (True Negative), and FN (False Negative). We overload these acronyms to be their corresponding numbers of samples, e.g., TP means the number of True Positive samples. Common classification performance metrics include the following.

- **Recall**, a.k.a. **TPR** (True Positive Rate): $\frac{TP}{TP+FN}$
- **FPR** (False Positive Rate): $\frac{FP}{FP+TN}$
- **Accuracy**: $\frac{TP+TN}{TP+TN+FP+FN}$
- **Precision**: $\frac{TP}{TP+FP}$
- **AUROC**: Area Under ROC (Receiver Operation Characteristic) curve

Among all ML classification performance metrics, we emphasize AUROC (Area Under ROC curve) and recall. Accuracy, precision, and F1 score results depend on the balance between positive and negative samples in data. For example, if the number of negative samples is much greater than the positive samples, even a low FPR can imply a large FP, which makes precision/accuracy very low. Recall, a.k.a. true positive rate (TPR), must be considered along with a certain false positive rate (FPR). AUROC is not sensitive to data imbalance and reflects the entire TPR-FPR tradeoff. Our goal is to find a technique with a large AUROC and fast inference.

In evaluating the ML model performance, we make sure that testing data are unseen in training. The "unseen" has two different scenarios:

- Relaxed split between training and testing data. Each testing sample is unseen in training data but may share the same Verilog description. For example, a placement solution of a JPEG decoder is a testing sample and not included in the training data. However, another logic synthesis and placement solution of the same JPEG Verilog description may exist in the training set.
- Strict split between training and testing data. If the placement solution of JPEG decoder is used as a testing sample, then there is no JPEG decoder at all in the training set.

III. Machine Learning Engines

In this work, we investigate four types of 6 different machine learning engines for the timing critical flip-flop prediction task.

A. Logistic Regression

Logistic regression [9] is a statistically supervised learning technique for estimating the probability that a data sample belongs to a class or not.

B. MLP (Multi-Layer Perceptron)

This is also known as artificial neural network [10]. In this work, binary cross-entropy is employed for MLP model training.

C. Decision Tree-Based Techniques

There are two popular decision tree-based techniques, random forest [11] and XGBoost [12]. Compared to neural network-based techniques, they are generally lightweight in terms of training and inference, and generally more interpretable.

1) Random Forest: A random forest consists of multiple decision trees, which are independently constructed with certain randomness. The final classification result is an assembled, e.g., voting, among these trees.

2) XGBoost: The XGBoost algorithm is implemented within the Gradient Boosting framework and provides a parallel tree boosting that is capable of fast and accurate training and inference for prediction tasks, and also curbs over-fitting by branch pruning.

D. Graph Neural Network

When GNNs are applied for classifications, there are two formulations. One is graph classification, which is to tell if an entire graph belongs to a class or not. The other is node classification, which decides if each node belongs to a class or not. For critical flip-flop prediction, each node represents a flip-flop and an edge implies combinational logic between two flip-flops. Evidently, the critical flip-flop prediction is node classification. We study two state-of-the-art GNN techniques: GraphSAGE [13] and Graph Attention Network [14].

1) GraphSAGE: Graph-based engines such as GraphSAGE ([13]) integrate the features of their neighbors encoded as embedding layers which is accomplished through the message-passing function and the aggregator function.

2) Graph Attention Network: Unlike GraphSAGE, which performs a simple formulation of the GNN layer and each neighbor contributes equally to the final representation of the target node, the Graph Attention Network ([14]) learns and assigns different importance to each neighbor's contribution.

IV. Machine Learning Model Features

The machine learning model features used in our method are described as follows.

- **Layout features**. There are two main layout features for our machine learning models. The first one is the longest fanin path length for each flip-flop. A path length is the total HPWL (Half Perimeter Wire Length) for all nets along a combinational logic path. The maximum HPWL of different timing paths is used, which is termed as (max distance) in the feature set. The other layout feature is the pin density (density) in a small region centered around the flip-flop to be classified.
- **Timing features**. These features are obtained from the cell library and post-placement (pre-CTS) timing report. Since it takes non-trivial time to obtain the timing report, the timing report-related features are optional and we evaluate models with and without them.
  - Setup time constraints. They vary for different flip-flop types.
  - Post-placement timing slack.
  - Flip-flop input slew rate.
  - The clock-to-output delay for a flip-flop.
  - Clock skew.
- **Structural features**. These are the fanout size (fanout), the maximum fanin logic depth (max logic depth), and the average fanin logic depth (avg logic depth) for the flip-flops to be classified.
- **Electrical feature**. This is the total capacitive load for each flip-flop to be classified.

For non-GNN engines, each data sample has one flip-flop with the specified feature set. For GNN techniques, an entire placement solution is abstracted to a graph, where each node corresponds to a flip-flop and edges indicate combinational paths among the flip-flops.

V. Data Preparation

Data plays a critical role in almost any machine learning approach. Data preparation includes data extraction, data processing, and splitting train and test cases. The work flow for data extraction is shown in Figure 2, which shows the major stages of a typical VLSI process with annotations on where the features and labels were extracted for this study. Preferably, we could evaluate timing feedback on flipflops as early as possible, and in this case, we chose to extract the features after the placement stage, and we used the slack obtained in the post-routing stage to generate the labels for each flipflop. The optimization step covers gate sizing and buffer insertion.

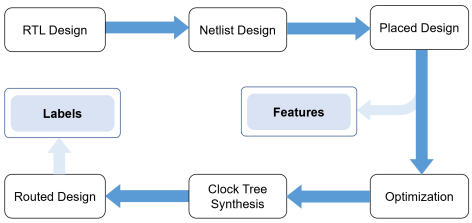

Fig. 2: Design flow for data preparation.

VI. Experiment

A. Experiment Setup

The benchmark designs with cell count and design runtime are summarized in Table I. Note that the number of combinational logic cells may change after the optimization. The "synthesis" here means CTS, optimization plus routing. There are 28 RTL designs. Each RTL design leads to multiple

placement solutions, each of which is performed with CTS, optimization, and routing. Overall, 148 routed designs are obtained. In the relaxed split scenario, these 148 routed designs are randomly divided into two sets, 106 samples (70% of total samples) for the training set and 42 samples for the testing set. The 106 training samples and 42 testing samples may share the same RTL designs. However, in the strict splitting scenario, 106 samples are used for training and 42 are used for testing, but the training and testing set in the strict splitting do not have overlapped RTL designs.

TABLE I: Benchmark circuits and their synthesis runtime

Design	Number of flip-flops	Total cell count	CTS (s)	Optimization (s)	Routing (s)	Total Synthesis (s)
jpegencode	39586	672641	3680	15310	4140	23130
tau17_leon2_iccad	149381	780456	12080	22438	16294	50812
gfx	49211	323269	3087	28876	3561	35524
scdma_viterbi	68393	316537	2977	2626	3992	9595
fft_256	42708	212402	2174	4198	2348	8720
fft_128	33461	172598	1595	11221	1722	14538
fft_64	16004	88129	830	6664	871	8365
ac97_ctrl	2199	4716	106	3487	126	3719
tate_pairing_911	86196	593296	9315	12539	12697	34551
tate_pairing_697	73144	512427	8045	10768	10966	29779
tate_pairing	31416	227656	3574	4784	4872	13230
des3_perf	8808	62247	977	1366	1332	3676
ethernet	10544	48616	763	1174	1040	2978
fpu	663	45854	720	962	981	2663
sd_card_controller	2646	12344	194	422	264	880
aes_core	530	11560	181	374	247	803
usb_funct	1744	11101	174	318	238	730
pci	2471	10516	165	381	225	771
trigonometric_functions	544	8218	129	210	176	515
systemcaes	670	5971	94	202	128	424
mem_ctrl	1065	5286	83	160	113	356
tv80	359	4382	69	125	94	287
pid_controller	396	3623	57	106	78	240
wb_dma	523	2904	46	103	62	211
des3_area	128	2546	40	138	54	233
spi	229	1773	28	98	38	164
systemcdes	190	1691	27	53	36	116
pwm	145	1372	22	121	29	172

The designs are based on 45nm Nangate OpenCell Library [15]. Placement solutions are obtained using Synopsys Design Compiler version S-2021.06-SP1. Clock Tree Synthesis and Routing are performed using Cadence Innovus version 191. The synthesis flow was run on a server with Intel(R) Xeon(R) CPU E5-2680 v2 @ 2.80GHz, 252GB RAM, CentOS Linux 7. Model training and inference are performed on Intel(R) Core(TM) i7-9850H @ 2.60GHz, 32GB RAM, Windows10 mobile workstation.

Random forest models are built with scikit-learn [16]. MLP is implemented with Keras [17]. XGBoost is implemented with the XGBoost python package [12]. StellarGraph [18] is used for graph representation for GraphSage with Tensorflow for the model pipeline. NetworkX [19] is used for graph representation for GAT with PyTorch used for the model pipeline. GraphSAGE uses BinaryCrossentropy as the loss function and GAT uses SigmoidCrossEntropy while both use the Adam optimizer.

The classification performance result for each RTL design is the average among all samples of the same RTL design.

B. Classification Performance

The average performance metrics and runtime among all test samples are summarized in Table II for all the machine learning engines evaluated in this work. One can see that XGBoost leads to the best AUC ROC (Area Under Curve of ROC) and recall. Although GAT obtains the best accuracy, its AUC ROC and recall values are inferior. Graph-based approaches also tend to incur large training and inference runtime. One main reason for the relatively low performance of GNN models is that the criticality of a flip-flop (or a graph node) only depends on its immediate incident edges, which correspond to the fanin and fanout combinational logic. As such, graph convolution among neighboring nodes (flip-flops) provides little help in the graph node classification. Our subsequent experimental results will be focused on XGBoost.

TABLE II: Classification performance of different ML engines

ML engines	AUC ROC	Accuracy	Recall FPR 5%	Training time (s)	Inference time (s)
Random Forest	0.96	99.6%	82.5%	2.1	0.03
XGBoost	**0.98**	**99.7%**	**94.4%**	5	0.31
MLP	0.94	99.6%	82.2%	105	0.08
Logistic Regression	0.88	99.6%	77.9%	27.1	0.01
GAT	0.88	99.9%	75.0%	5,820	5.23
GraphSage	0.87	88.0%	67.5%	11,502	50.42

The classification performance and runtime of of our XGBoost-based models are shown in Table III for individual designs. The Average speedup is the ratio of the average total synthesis time to the average model inference time. Compared to running a synthesis flow of CTS, optimization, and routing, our models achieve a speedup of more than $62K\times$. We further compare the ROC curves of our XGBoost-based and MLP-based models in Figure 3, where the advantage of XGBoost is evident.

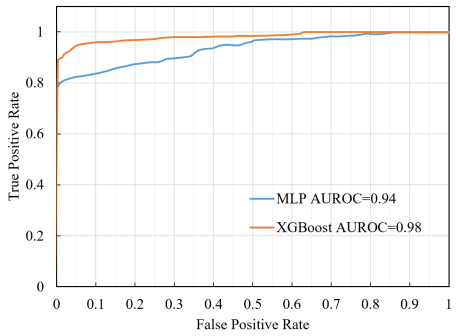

Fig. 3: ROC curves for our XGBoost and MLP models

C. Comparison with Previous Work

To the best of our knowledge, our work is the first one on critical flip-flop prediction. We compare with the state-of-the-art timing prediction work [8], which can be applied for critical flip-flop prediction with the assumption that there is no optimization performed with netlist change. We find a few designs where this assumption can be satisfied and compare our XGBoost-based method with [8] on these designs. The results are summarized in Table IV. The classification performance of our model is similar to [8], with better accuracy and slightly worse recall/FPR. Our inference time is nearly $8\times$ faster than [8].

TABLE III: Performance and runtime of our XGBoost-based model with full feature set

Design	AUROC	Accuracy	Recall (FPR = 5%)	Inference time (s)	Speedup vs total synthesis
ac97_ctrl	0.9	99.0%	100%	0.031	119,009×
fft_64	1	99.4%	100%	0.141	59,465×
fft_128	1	99.9%	100%	0.218	66,622×
fft_256	0.98	99.9%	93.2%	0.304	28,689×
gfx	1	99.9%	100%	0.322	110,471×
scdma_viterbi	0.99	99.6%	100%	0.250	38,420×
jpegencode	0.96	99.9%	83.0%	0.273	84,664×
tau17_leon2_iccad	0.98	99.9%	79.0%	0.937	54,200×
Average	0.98	99.7%	94.4%	0.310	62,358×

TABLE IV: Comparison between our XGBoost-based model and [8]

Design	Accuracy		Recall		FPR		Inference time(s)		Speedup vs [8]
	Our XGBoost	[8]	Our XGBoost	[8]	Our XGBoost	[8]	Our XGBoost	[8]	
fft_128	99.9%	94.4%	91.1%	95.7%	5.0%	3.5%	0.229	2.201	9.6×
fft_256	99.9%	94.9%	89.0%	95.3%	5.0%	2.9%	0.314	2.214	7.1×
jpegencode	99.9%	95.5%	90.0%	95.8%	5.0%	2.6%	0.281	1.932	6.9×
Average	99.9%	94.9%	90.0%	95.6%	5.0%	3.0%	0.275	2.116	7.8×

D. Relaxed and Strict Split between Training and Test Data

According to the splitting schemes described in Section II, the performance of our XGBoost-based models shown in Table V suggests that our XGBoost models work pretty well even under the strict split, where the test designs are completely unseen in the training.

TABLE V: Comparison between different training/test split schemes

Design	AUROC		Accuracy		Recall (FPR = 5%)	
	Relaxed	Strict	Relaxed	Strict	Relaxed	Strict
fft_128	1	0.97	99.9%	99.9%	100.0%	91.1%
fft_256	0.98	0.95	99.9%	99.9%	93.2%	89.0%
jpegencode	0.96	0.96	99.9%	99.9%	83.0%	90.0%
Average	0.98	0.96	99.9%	99.9%	92.1%	90.0%

E. Impact of Slack Thresholds

The timing criticality of a flip-flop depends on a user-specified threshold. One can choose the threshold to be 0 slack, which means flip-flops with negative slacks are critical. Alternatively, one can choose a higher threshold, e.g., $20ps$, which implies that all flip-flops with slacks below $20ps$ are treated as critical. In Table VI, we show the results of our XGBoost model on the same test cases as Table III for different threshold values. The AUC ROC and accuracy of our models are not sensitive to threshold changes. Since recall values are associated with specific FPR, i.e., specific points on a ROC curve, they are subject to local variations.

F. Impact of Feature Selections

The full feature set described in Section IV has 17 features and includes those from post-placement (pre-CTS) timing analysis. Since the timing analysis time is non-trivial, we consider a reduced feature set, where the timing analysis-related features are removed and the other 12 features remain. The classification performance and runtime of our XGBoost models with the reduced feature set are evaluated and the results are shown in Table VII. Comparing with results from the full feature set in Table III, one can see that the classification performance is similar while the inference time is shorter.

To further study the impact of individual features, we obtain the feature importance rankings for the full set and the reduced

TABLE VI: Classification performance under different slack thresholds

Threshold (ns)	AUROC	Accuracy	Recall
0	0.97	99.2%	87.2%
0.01	0.98	99.7%	94.4%
0.02	0.96	99.8%	88.8%
0.03	0.97	99.7%	90.2%
0.04	0.95	98.4%	91.4%

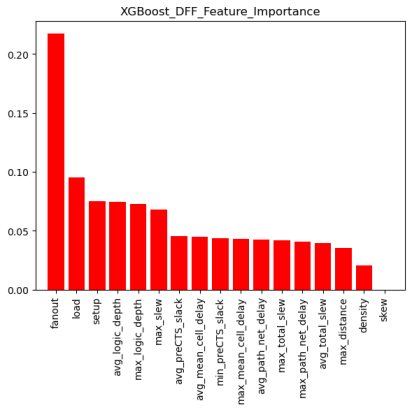

Fig. 4: Feature importance ranking for the full feature set.

TABLE VII: Classification performance and inference time of our XGBoost models with a reduced feature set

Design	AUROC	Accuracy	Recall (FPR=5%)	Inference time (s)	Speedup vs total synthesis
ac97_ctrl	0.98	99.8%	100%	0.016	238,397×
fft_64	0.95	98.9%	95.6%	0.078	107,244×
fft_128	0.97	99.9%	92.2%	0.156	93,192×
fft_256	0.98	99.9%	86.5%	0.156	55,897×
gfx	0.97	99.8%	100%	0.208	170,788×
scdma_viterbi	0.99	99.6%	100%	0.389	24,666×
jpegencode	0.96	99.7%	85.3%	0.179	129,218×
tau17_leon2_iccad	0.93	99.6%	69.5%	0.921	55,170×
Average	0.97	99.7%	91.1%	0.263	73,434×

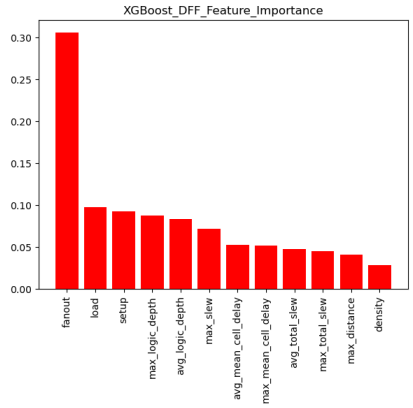

Fig. 5: Feature importance ranking for the reduced feature set.

set, which are depicted in Figures 4 and 5, respectively. These two figures show only the features of top importance. There is a large overlap between the two figures and this tells that our method has little dependence on the timing analysis-based features.

VII. Conclusion

This paper explored how machine learning can be used to predict timing critical flip-flops in the context of post-placement. The major work accomplished in this study include proposing a machine learning flow based on the XGBoost technique, identification of top features that are correlated with flip-flop timing criticality, and construction of a dataset based on 28 benchmarks for training and testing. Results show that the machine learning flow is useful for obtaining high prediction accuracy and recall while being general enough to be used on new test designs. Future work may integrate the flow to assist timing-driven placement.

Acknowledgment

This work is partially supported by Semiconductor Research Corporation GRC-CADT 3103.001/3104.001 and National Science Foundation CCF-2106725/2106828.

References

[1] D. A. Papa, T. Luo, M. D. Moffitt, C. N. Sze, Z. Li, G.-J. Nam, C. J. Alpert, and I. L. Markov, "Rumble: an incremental, timing-driven, physical-synthesis optimization algorithm," *Proceedings of the 2008 international symposium on Physical design*, pp. 2–9, 2008.

[2] J. Jung, G.-J. Nam, W. Chung, and Y. Shin, "Integrated latch placement and cloning for timing optimization," *ACM Transactions on Design Automation of Electronic Systems*, vol. 24, no. 2, pp. 1–17, 2019.

[3] C.-C. Huang, G. Tellez, G.-J. Nam, and Y.-W. Chang, "Latch clustering for timing-power co-optimization," *Proceedings of the 57th ACM/IEEE Design Automation Conference*, pp. 1–6, 2020.

[4] D. Mangiras, A. Stefanidis, I. Seitanidis, C. Nicopoulos, and G. Dimitrakopoulos, "Timing-driven placement optimization facilitated by timing-compatibility flip-flop clustering," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 39, no. 10, pp. 2835–2848, 2019.

[5] E. C. Barboza, N. Shukla, Y. Chen, and J. Hu, "Machine learning-based pre-routing timing prediction with reduced pessimism," *Proceedings of the 56th ACM/IEEE Design Automation Conference*, pp. 1–6, 2019.

[6] T. Yang, G. He, and P. Cao, "Pre-routing path delay estimation based on transformer and residual framework," *Proceedings of the 27th Asia and South Pacific Design Automation Conference*, pp. 184–189, 2022.

[7] L.-W. Chen, Y.-N. Sui, T.-C. Lee, Y.-L. Li, M. C.-T. Chao, I.-C. Tsai, T.-W. Kung, E.-C. Liu, and Y.-C. Chang, "Path-based pre-routing timing prediction for modern very large-scale integration designs," *Proceedings of the 23rd IEEE International Symposium on Quality Electronic Design*, pp. 1–6, 2022.

[8] Z. Guo, M. Liu, J. Gu, S. Zhang, D. Z. Pan, and Y. Lin, "A timing engine inspired graph neural network model for pre-routing slack prediction," *Proceedings of the 59th ACM/IEEE Design Automation Conference*, pp. 1207–1212, 2022.

[9] L. Connelly, "Logistic regression," *Medsurg Nursing*, vol. 29, no. 5, pp. 353–354, 2020.

[10] S.-C. Wang, "Artificial neural network," *Interdisciplinary Computing in Java Programming*, pp. 81–100, 2003.

[11] J. L. Speiser, M. E. Miller, J. Tooze, and E. Ip, "A comparison of random forest variable selection methods for classification prediction modeling," *Expert Systems with Applications*, vol. 134, pp. 93–101, 2019.

[12] T. Chen and C. Guestrin, "Xgboost: A scalable tree boosting system," *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 785–794, 2016.

[13] W. Hamilton, Z. Ying, and J. Leskovec, "Inductive representation learning on large graphs," *Proceedings of the 31st Conference on Neural Information Processing Systems*, vol. 30, 2017.

[14] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Lio, and Y. Bengio, "Graph attention networks," *arXiv*, 2017.

[15] Silvaco, "45nm open cell library."

[16] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg *et al.*, "Scikit-learn: Machine learning in python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[17] F. Chollet *et al.*, "Keras," https://keras.io, 2015.

[18] C. Data61, "Stellargraph machine learning library," https://github.com/stellargraph/stellargraph, 2018.

[19] D. A. S. Aric A. Hagberg and P. J. Swart, "Exploring network structure, dynamics, and function using networkx," *Proceedings of the 7th Python in Science Conference*, pp. 11–15, 2008.