Machine-Learning Based Delay Prediction for FPGA Technology Mapping

Hailiang Hu, Jiang Hu
Texas A&M University
College Station, TX, USA
hailiang@tamu.edu, jianghu@tamu.edu

Fan Zhang, Bing Tian, Ismail Bustany
Advanced Micro Devices
San Jose, CA, USA
zhangf2@amd.com, bing.tian@amd.com,
ismail.bustany@amd.com

ABSTRACT

Accurate delay prediction is important in the early stages of logic and high-level synthesis. In technology mapping for field programmable gate array (FPGA), a gate-level circuit is transcribed into a lookup table (LUT)-level circuit. Quick timing analysis is necessary on a pre-mapped circuit to guide optimizations downstream. However, a static timing analyzer is too slow due to its complexity and highly inaccurate like other faster empirical heuristics before technology mapping. In this work, we present a machine learning based framework for accurately and efficiently estimating the delay of a gate-level circuit from predicting the depth of the corresponding LUT logic after technology mapping. Our experimental results show that the proposed method achieves a **56x** accuracy improvement compared to the existing delay estimation heuristic. Instead of running the mapper for the ground truth, our delay estimator saves **87.5%** on runtime with negligible error.

CCS CONCEPTS

Hardware → EDA → Logic synthesis → Technology mapping;

Computing methodologies → Machine learning → Machine learning algorithms

KEYWORDS

FPGA; Synthesis; Delay Estimation; Technology Mapping; Machine Learning

1 Introduction

Early decisions in high-level synthesis or logic synthesis greatly impact the final timing quality of result (QoR). However, delay estimation is often too coarse at that stage to guide optimizations. The key reason is that the design is often represented with a high level graph or technology-independent gates and not mapped into FPGA hardware primitives.

The operation node in the control dataflow graph (CDFG) in high-level synthesis and the dataflow graph (DFG) in logic synthesis can be lowered and generated as a technology-independent a gate-level netlist, which is also input to the Boolean optimization engine. Our work chooses gate-level netlist as an anchor point and presents a machine learning (ML)-based method to estimate delay on pre-mapped gate-level netlist. The method is 8x faster than running FPGA technology mapping and has over 99% accuracy score with the mean squared error of less than 2 logic levels.

The remainder of the paper is organized as follows. In Section 1.1, the process of FPGA technology mapping is presented. Section 1.2 provides an overview of ML in electronic design automation (EDA). Our ML framework is described in section 2. Section 3 discusses how the proposed framework can be integrated into the logic synthesis and high-level synthesis flows. The experimental setup and detailed results and analyses are presented in section 4. Section 5 presents our conclusions and future work.

1.1 Technology Mapping in FPGA Synthesis

A *Boolean network* is a directed acyclic graph (DAG) with nodes corresponding to logic gates and directed edges corresponding to connections between them. A combinational logic function can be represented with a Boolean network noncanonically. One logic function can have many representations in a Boolean network generated by Boolean optimizations or by construction. Technology mapping in FPGA synthesis is the procedure of transforming a Boolean network into a network of lookup tables (LUTs). A LUT can have up to k inputs and can implement as many as 2^{2^k} combinational logic functions. Figure 1 shows a simplified gate-level Boolean network and its post-mapping result.

SLIP '22, November 3, 2022, San Diego, CA, USA

ACM ISBN 978-1-4503-9536-6/22/11…$15.00
https://doi.org/10.1145/3557988.3569713

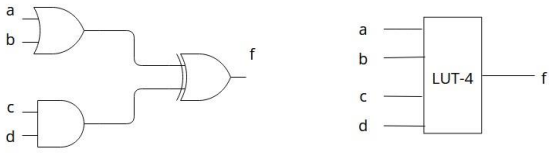

Figure 1: A Boolean network representing $f = (a|b) \oplus (c\&d)$ on the left and its LUT implementation after technology mapping on the right.

A Boolean network (also referred to as gate-level netlist in our context) for real combinational circuits can be large and has a complex structure with re-convergent fanouts. A standard technique for mapping a Boolean network into k-input LUTs is to formulate it as a graph covering the problem [1,2,3]. Cuts are computed on each node to cover logic cones in the Boolean network with k-input LUTs. Costing is associated with each candidate cut and propagated from input to output. After the cut computation, a backward pass is executed to select the best cut choice. Most of the runtime is spent during the cut computation phase, with a runtime complexity of $O(n^k)$, where n is the number of nodes and k is number of LUT inputs [4]. Depending on different cost functions (delay, area, and wire length) and the topology of the pre-mapped Boolean network, the generated LUTs circuitry can have a wide range of logic levels and LUT count. Due to this variety, it is desirable to predict the delay of the mapped result prior to mapping, where we believe ML is an effective technique.

1.2 ML in EDA

In recent years there has been burgeoning research on applying ML in EDA. For logic synthesis, Keren et al. adopted Graph Neural Network (GNN) and Reinforcement learning (RL) to find the sequence of optimizations to reduce logic delay and gate count [5]. Rai et al. present learning incompletely-specified functions on the results of the recent IWLS 2020 competition [15]. Zhu et al. proposed a Markov decision process formulation for the logic optimization problem and RL approach incorporating a graph convolutional network to explore the solution search space. Their empirical results show improvements over well-known logic heuristics [16]. For the backend, Mirhoseini et al. demonstrated a RL-based macro placer on TensorFlow ASIC blocks [14]. Ghandi et al. proposed a RL based router to route circuitry and fix violations with limited access to labeled data [6]. Baig et al. proposed an RL-based detailed routing approach for FPGA detailed routing attaining a 35% speedup with similar or better quality of results [13]. A whole class of regression or classification techniques have been used for prediction. For example, Random Forest is applied to predict timing delay during placement with 94% accuracy [7]. Maarouf et al. created a competitive Linear Regression model with comparable accuracy but on average 291x faster than KNN and MLP based models to predict FPGA routing congestion [8]. Elgammal et al. [11] enhanced a SA-based FPGA placer with RL and targeted perturbations and showed 2.5x speedup with comparable quality to VTR 8 [12].

To our knowledge, there is no prior ML-based research work to predict timing delay before technology mapping. Delay is usually estimated with pre-built libraries, which do not scale well and suffer from inaccurate heuristics.

2 Proposed Framework

2.1 Overview

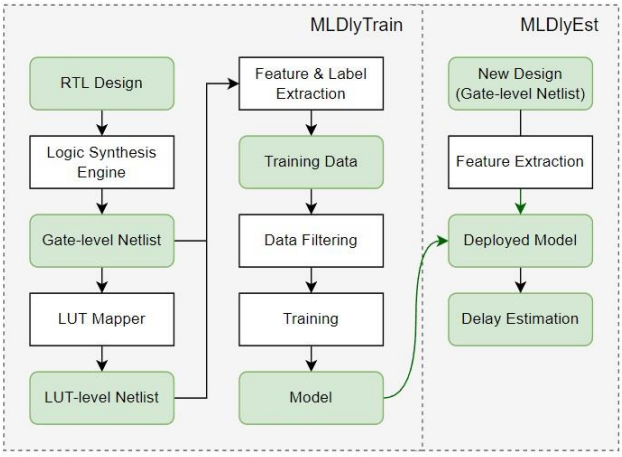

Figure 2: Our proposed framework, MLDlyTrain for training and MLDlyEst for inference.

The proposed framework comprises two parts, shown in Figure 2. The first part is MLDlyTrain, which does the feature and label extraction on a pool of designs and trains the ML model to predict post-mapped LUT level. The second part is MLDlyEst for logic depth inference. The LUT logic level based delay estimate is accurate enough for logic and high-level synthesis.

2.2 Features and Label Extraction

Twelve features in four categories have been selected to capture the topology of the gate-level netlist and behavior of technology mapping:

1) Number of Primary Inputs (PIs), in general reflects the scale of a circuit, which helps make some quick decisions (e.g., any circuit with six or fewer primary inputs can be encoded into a single LUT6).

2) Number of Gates is another metric of the size of a circuit. To better describe the circuits and capture the nature of LUT mapping, we categorize gates by their associated number of PIs (e.g., Gates with the number of PI ∈ (0,6], (6,12], (12, 18], …, (31, +∞)) and keep track of the number of gates in each category.

3) Number of Paths is a good supplement to the two features above on the complexity of a circuit. It helps identify the circuit's fanout and indicates the circuit's wideness.

4) Path Length in a circuit gives an estimation of the depth of a circuit. We use four features to measure the length: (a) total

path length, (b) average path length, (c) maximum path length and (d) minimum path length.

The combination of the number of paths and path length sketches the shape of a circuit on wideness and depth. A depth-first search is performed on the gate-level netlist to extract the features. Our traversal algorithm consumes only $O(n)$ runtime, where n is the number of cells in the circuit. The process of extracting labels shares the same framework as extracting features, but instead of traversing the gate-level circuit, it traverses the LUT-level circuit and captures LUT logic levels.

Once the features and labels are acquired and combined as training data, they are then filtered to remove abnormal and trivial cases. For example, a large netlist could have too many paths leading to integer overflow, and the number needs to be converted to a floating point. Trivial cases like constant logic, which can lead to 0-level, are removed. The whole dataset is then divided into 80% for training and 20% for testing.

2.3 Proposed ML Model

In this work, we adopt a supervised learning model based on gradient boosting, Light Gradient Boosting Machine (LGBM) [9] as it works well on the highly non-linear relationships that underly delay estimation. Our framework can be easily integrated into AMD's Vivado ML®Compiler.

3 Flow Integration

After the model is trained using the proposed framework, it can be applied at various stages in logic synthesis and high-level synthesis. Figure 3 and 4 show how it can be integrated in high-level synthesis during scheduling and logic synthesis for operator sharing.

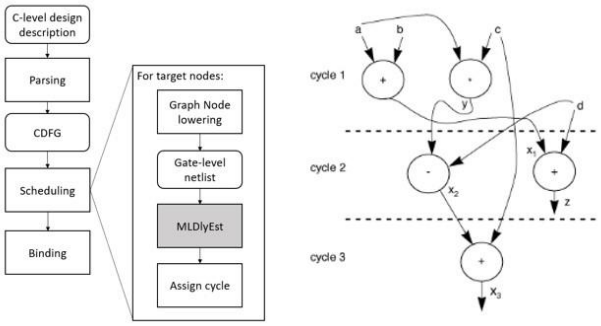

Figure 3: Delay prediction integration into high-level synthesis(left). Nodes in CDFG assigned cycles after predicting delay of nodes(right)

In high-level synthesis, during scheduling, operation nodes in CDFG are assigned cycles, and the delay of operations cannot exceed the clock period times the number of cycles. Similarly, in logic synthesis, DFG optimization, such as operator sharing, makes delay-area trade-offs based on early estimates. In both cases, nodes in DFG and CDFG can be lowered into gate-level netlist, and MLDlyEst can be applied to provide accurate delay estimation. In Figure 5, corresponding LUT level delays are predicted respectively for the adder and comparator. They are simply added up and compared with a timing budget, which can be derived from design timing constraints or any heuristics. If the total delay is not critical, then operator sharing can be applied to save the area of one adder.

Figure 4 also shows how MLDlyEst can be integrated into the Boolean optimization engine to choose a better implementation for FPGA mapping from candidates generated from different algorithms.

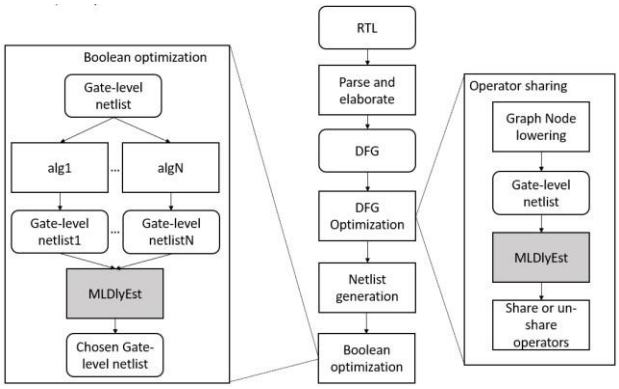

Figure 4: Delay prediction integration into early DFG and Boolean optimization stage of logic synthesis

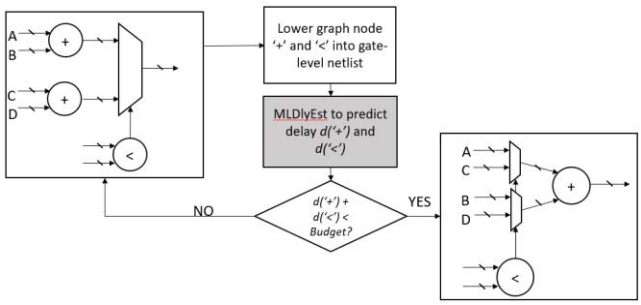

Figure 5: Operator sharing guided by prediction from MLDlyEst

MLDlyEst is easy to integrate, and there is no need to change the feature set nor add customized features to retrain the ML model for different tools since the trained model works on a uniform gate-level netlist.

4 Experimental Results

This section shows the evaluation of models based on accuracy and runtime metrics. The experiment is conducted with a Linux Machine running on a Xeon 2.6GHz processor. All ML models are

implemented with Scikit-learn [10]. Logical synthesis and technology mapping is carried out by the Vivado ® Synthesis Tool.

4.1 Benchmark Design Suite

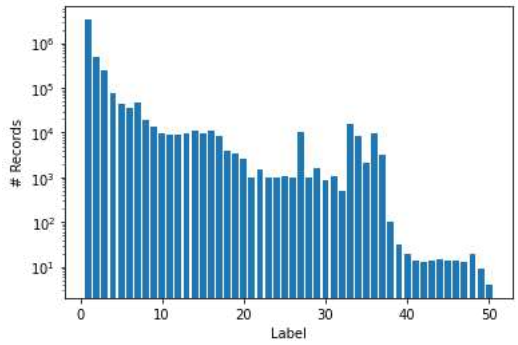

Figure 6: Number of Records for Each Label (Log Scale)

Our benchmark design suite contains 86 RTL industrial designs. Each design, on average, contains 50K sub-circuits used as data points. The Vivado Synthesis ® tool is invoked to transfer the RTL design into the gate-level netlist, and then the LUT-level netlist.

The histogram in Figure 6 shows the distribution of the labels on the benchmark suite designs. It is evident that the dataset is heavily unbalanced (94% of the sampled labels have depth less than 5). Clearly, special handling is needed for the minority class to ensure the performance on all records.

4.2 Model Training

As mentioned in Section 2.2, data is split randomly into 80% training and 20% testing sets. Cross-validation is done to ensure the stability of our model. Different options in model training affect the performance of the model. In this section, we list the options that work best for our cases.

In general, increasing the number of estimators helps reduce the prediction errors without a significant increase on the inference runtime. In our model, the number of estimators is set to 1000. The number of estimators, maximum tree depths, step length and other parameters can be optimized through an automatic hyperparameter tuner, which is our intention for future work.

As shown in Figure 6, the distribution of the labels in our benchmark is unbalanced, which could make the focus of the training process on the majority, thus giving poor results on the under-represented classes. To mitigate this problem, we increase the LGBM weights for the less frequent data. In particular, we set the weight to be inversely proportional to the class frequencies during training.

4.3 Error of LGBM on Depth and Area Driven Mapper

Table 1 shows the performance of ML model in predicting the result of technology mapping when the mapper runs in different modes. For the depth-driven mode, the mapper tries to minimize the depth of the mapped netlist. For the area-driven mode, the mapper optimizes the depth and the area of the design at the same time. The experimental results shown are generated from the test dataset.

Since the dataset is unbalanced, we evaluate the model on different ranges of labels separately. By utilizing the weight adjustment technique mentioned in Section 4.2, we achieve similar accuracy among the different label classes.

From the column of mean absolute error (MAE), one can tell that our model predicts the LUT depth well. In most label classes, the MAE is less or equal to 0.2 and 0.5 for the depth and area-driven mapper respectively. Even though the relative percentage error is not explicitly given, one can infer that it is upper bounded by 20%, which appears to be the MAE for the area-driven mapper for label class less than or equal to five.

In addition to MAE, two more metrics are presented to help understand the prediction result: "Err 99.7%" is the upper bound error for 99.7% of the records. For example, Column 3 shows that 99.7% of all the records with LUT depth less or equal to five have errors less or equal to 1.31. Also, in Table 1, "Err<=2 Pct" is the percentage of records that have a prediction error less than or equal to two logic levels.

Label	MAE		Err 99.7%		Err<=2 Pct	
	Depth	Area	Depth	Area	Depth	Area
[1,50]	0.09	0.23	1.31	2.50	100.00%	99.62%
[1, 5]	0.09	0.20	1.31	2.21	100.00%	99.82%
[6,10]	0.14	0.63	1.38	5.38	99.98%	96.26%
[11,20]	0.11	0.50	1.80	3.93	99.88%	96.48%
[21,30]	0.05	0.32	0.67	1.71	99.98%	99.25%
[31,40]	0.17	0.27	1.27	1.27	99.96%	99.78%
[41,50]	0.54	1.36	1.15	1.69	100.00%	96.25%

Table 1: LGBM Prediction Accuracy on Depth-driven and Area-driven Mapper

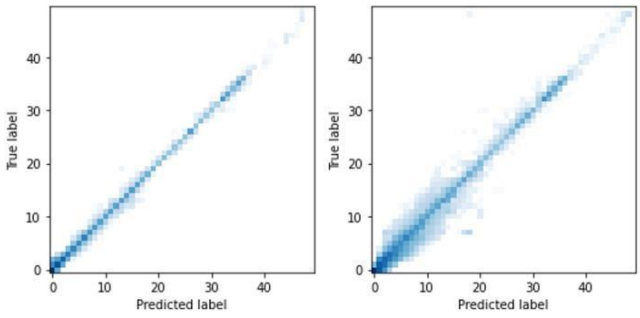

Figure 7: Confusion Matrix of the LGBM Model w.r.t. Depth (left) and Area(right) Driven Mapper

The metrics in Table 1 show that the ML model performs better on the depth-driven mapper compared to the area-driven mapper. This observation coincides with our expectations as the area-driven mapper enables resource sharing among the neighboring sub-circuits, which is not well captured in our feature extraction process. The confusion matrix in Figure 7 shows the same trend.

4.4 Runtime Analysis

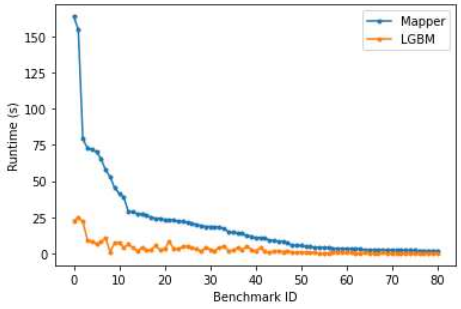

Figure 8: Mapper and LGBM ML Model-based Delay Estimation Runtime (s)

Figure 8 shows the runtime of the mapper and the ML model over the benchmarks. On average, The ML model is 8x faster than the mapper. The speedup varies depending on the scale of the netlist. The benchmarks on the left-hand side (i.e. with lower ID's) are designs with large and deep combinational logic. Their respective runtime gain is significantly higher than the designs with less combinational logic on the right-hand side. This observation supports the runtime complexity analysis of $O(n)$ for the ML model vs. $O(n^k)$ for the mapper alluded to earlier. Mapper's runtime is of much higher order because it needs to enumerate Cuts for each node. Rather than the highly pipelined design where shallow netlists dominate, our model shows more potential on a deep netlist or untimed large logic cones, which, in practice, is more frequently the case when a prediction is needed.

4.5 Importance of Features

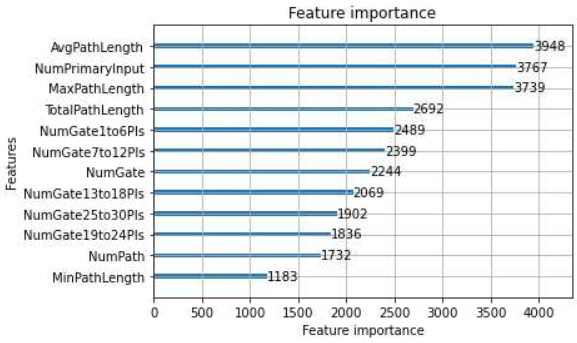

Figure 9: Feature Importance of the LGBM ML Model

Figure 9 shows the importance of features in the proposed model. The number of PIs and the path length information dominate, while the number of gates with different PIs also plays an important role.

4.6 Error of Alternative Models and the Timer: A Comparison

Model	MAE	MSE	Accuracy
LGBM	0.09	0.04	96.02%
Timer	5.03	288	0.03%
LR	0.26	0.38	83.30%
MLP	0.59	0.06	81.78%

Table 2: Error of Timer and Models

Linear regression (LR) and multi-layer perceptron (MLP) are implemented to explore the performance of different models. Besides the features mentioned in section 2.2, we added the logarithm of the number of paths and total path length with a base of two, which exposes the linear characteristics from the exponentially growing values. We tuned these models for a fair comparison. The performance of a pre-mapping timer is also listed for comparison. Similar to the post-mapped timer, the pre-mapped timer computes arrival time based on the delays associated with each pre-mapped gate and net. Considering that there is much larger number of pre-mapped gates and nets compared to post-mapped, their associated delays are adjusted and dampened. The pre-mapped timer gives an albeit rough but acceptable estimate on delay.

Table 2 shows the prediction results of Timer and other models on the testing data. The computed arrival time from Timer is normalized to LUT levels for comparison against ML models. LGBM is superior to other models on MAE and mean squared error (MSE). Accuracy in the table is defined as the 'hit rate' of predictions. For example, if the ground truth is 5 LUT levels and the predicted value is 4.4 (rounded down to 4), it is counted as miss. And if the predicted value is 4.6 (rounded up to 5), it is counted as a hit. Although Timer can give a rough estimate, it has a very low hit rate because it tries to use an overly simplified axis (critical path arrival time) to predict a much harder problem. LR gives better performance than Timer since it takes more features than just critical path depth. LGBM outperforms the rest because not only it has more features, but it can also model non-linear behavior. Compared to Timer, the integration of LGBM ML model yields a big leap of 56x accuracy (MAE ratio) improvement on delay estimation in the early stages of synthesis.

5 Conclusions

In this paper, a machine learning based framework for delay estimation on gate-level netlist is presented. The LGBM model gives an estimation with an error of less than 2 for most of the

benchmarks with 8x faster runtime compared with the depth-driven mapper. Due to its flexibility, our model can be deployed in multiple stages to guide the netlist optimization in the synthesis flow. Our future work will focus on improving the performance of the estimation on the area-driven mapper and predicting the delay of specific path in the netlist.

REFERENCES

[1] J. Cong and Y. Ding, "FlowMap: An optimal technology mapping algorithm for delay optimization in lookup-table based FPGA designs," IEEE Trans. CAD, Vol.13(1), Jan. 1994, pp. 1-12, doi: 10.1109/43.273754.

[2] J. Cong and Y. Ding, "On area/depth trade-off in LUT-based FPGA technology mapping," IEEE Trans. VLSI, Vol 2(2), Jun. 1994, pp. 137-148, doi: 10.1109/92.285741.

[3] J. Cong, C. Wu and Y. Ding, "Cut ranking and pruning: Enabling a general and efficient FPGA mapping solution," Proc. FPGA 99, pp. 29-36.

[4] R. Brayton, S. Chatterjee, A. Mishchenko, "Improvements to Technology Mapping for LUT-Based FPGAs," IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 26, no. 2, pp. 240-253, Feb. 2007, doi: 10.1109/TCAD.2006.887925.

[5] K. Zhu, M. Liu, H. Chen, Z. Zhao and D. Pan, "Exploring Logic Optimizations with Reinforcement Learning and Graph Convolutional Network," 2020 ACM/IEEE 2nd Workshop on Machine Learning for CAD (MLCAD), 2020, pp. 145-150, doi: 10.1145/3380446.3430622.

[6] U. Gandhi, I. Bustany, W. Swartz and L. Behjat, "A Reinforcement Learning-Based Framework for Solving Physical Design Routing Problem in the Absence of Large Test Sets," 2019 ACM/IEEE 1st Workshop on Machine Learning for CAD (MLCAD), 2019, pp. 1-6, doi: 10.1109/MLCAD48534.2019.9142109.

[7] T. Martin, G. Grewal and S. Areibi, "A Machine Learning Approach to Predict Timing Delays During FPGA Placement," 2021 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW), 2021, pp. 124-127, doi: 10.1109/IPDPSW52791.2021.00026.

[8] D. Maarouf et al, "Machine-Learning Based Congestion Estimation for Modern FPGAs," 2018 International Conference on Field-Programmable Logic and Applications (FPL), 2018, pp. 427-4277, doi: 10.1109/FPL.2018.00079.

[9] G. Ke et al, "LightGBM: a highly efficient gradient boosting decision tree," In Proceedings of the 31st International Conference on Neural Information Processing Systems (NIPS'17). Curran Associates Inc., Red Hook, NY, USA, 3149–3157, doi: 10.5555/3294996.3295074.

[10] Pedregosa et al., "Scikit-learn: Machine Learning in Python," 2011 JMLR 12, pp. 2825-2830.

[11] M. A. Elgammal, K. E. Murray, V. Betz, "RLPlace: Using Reinforcement Learning and Smart Perturbations to Optimize FPGA Placement," IEE Transactions on Computer-Aided Design and Integrated Circuits and Systems, no. 8 (2022): 2532-2545 , doi: 10.1109/TCAD.2021.3109863.

[12] K. E. Murray et al. "VTR 8: High Performance CAD and Customizable FPGA Architecture modelling," ACM Transactions on Reconfigurable Technologies and Systems, vol. 13, no. 2, pp. 1-55, 2020, doi: 10.1145/3388617.

[13] I. Baig, U. Farooq, "Efficient Detailed Routing for FPGA Back-End Flow Using Reinforcement Learning," Electronics 11 (14)2240, 2022, doi: 10.3390/electronics11142240.

[14] A. mirhoseini, A. Goldie, M. Yazgan, J. Jiang, E. Songhori, S. Wang, Y.-J. Lee, E. Johnson, O. Pathak, S. Bae et al. "Chip Placement With Deep Reinforcement Learning," arXiv preprint arXiv:2004.10746, 2020.

[15] S. Rai, W. :. Neto, Y. Miyasaka, X. Zhang, M. Yu, Q. Yi, et al, "Logic Synthesis Meets Machine Learning: Trading Exactness for Generalization," 2021 Design, Automation & Test in Europe Conference & Exhibition (DATE), 2021, pp. 1026-1031, doi: 10.23919/DATE51398.2021.9473972.

[16] K. Zhu, M. Liu, H. Chen, Z. Zhao, D. Pan, "Exploring Logic Optimizations with Reinforcement Learning and Graph Convolutional Network," 2020 ACM/IEEE 2nd Workshop on Machine Learning for CAD (MLCAD), 2022, pp. 145-150, doi: 10.1145/3380446.3430622.