ML-based Routing Congestion and Delay Estimation in Vivado ML Edition

WP532 (v1.0) October 26, 2021

Abstract

The FPGA physical design flow offers a compelling opportunity for Machine Learning for CAD (MLCAD) for the following reasons:

- An ML solution can be applied wholesale to a device family.
- There is a vast data farm that can be harvested from device models and design data from broad applications.
- There is a single streamlined design flow that an be instrumented, annotated, and queried at all stages.

In this white paper, two ML modeling applications are provided to enhance the accuracy of timing delay and routing congestion estimation in the Vivado® ML edition. Accurate delay estimation is pertinent for timing closure because the global/detail placer, physical synthesis in placement, and the global router use it to estimate net criticalities. The ML-based delay estimator improves the accuracy from 65.5% to about 98%. The Vivado Design Suite placement flow relies on a routing congestion estimator to identify and alleviate routing congestion hotspots during placement so that the design is easier to route downstream. The ML-based congestion estimator in the Vivado ML edition demonstrates similarly significant accuracy gains over the traditional approaches, and results in marked routing run-time reductions on a broad suite of designs from various device families.

Introduction

MLCAD is used in the ASIC physical design (PD) flow to solve complex computational problems because it can:

- Identify trends, patterns, and clusters of data.
- Handle multiple dimensions and multiple variables.
- Model highly nonlinear behavior.
- Work efficiently on large data sets.

Like ASICs, the FPGA PD flow aims to formulate and efficiently solve a discrete, multi-objective, non-linear optimization problem that maps a design onto a physical layout with interconnect logic to achieve target performance metrics. However, there are key differences between the two that underscore the reasons why MLCAD is even more compelling to the FPGA PD flow.

Unlike ASICs, FPGA place and route is implemented on a layout template with interconnected routing resources. This type of logic mapping can be challenging because the number and type of available logic and routing resources are fixed. However, FPGAs provide an ideal framework for using ML solutions for entire device families to improve device modeling accuracy and speed up the convergence of various algorithms in the PD flow. Some of the reasons MLCAD is ideal for FPGAs are listed in this section.

For a good survey on the application of ML in FPGA PDs, see *Machine Learning for Electronic Design Automation: A Survey* [REF 1].

Reason 1: FPGAs are Templatized Design Platforms

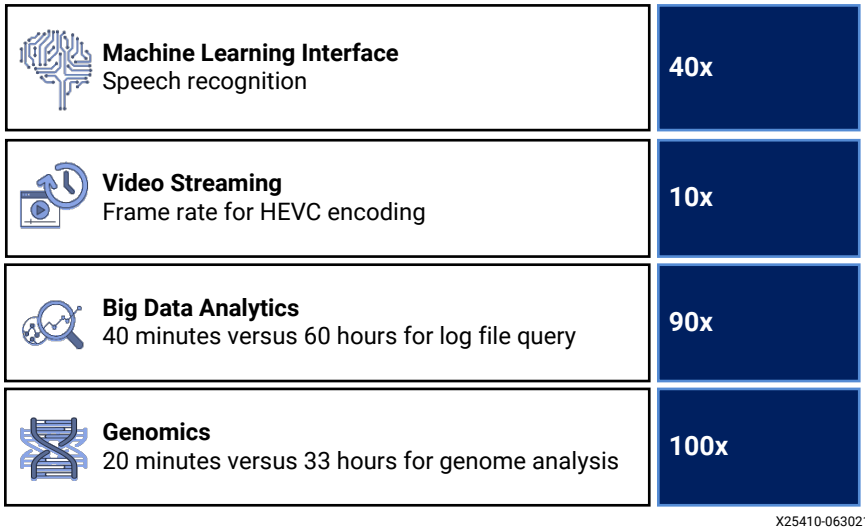
FPGA families are template-based programmable platforms with the following features:

- Serve a wide range of applications.
- Long life cycle.
- Interconnect and logic layout features, captured by detailed device models.
- Uniform device characteristics per device family.
- Predictable regular layout.
- Simpler timing and routing models (buffered interconnect and 2D abstracted routing layers).
    - ML effort has a high ROI because it applies to an entire device family.

Reason 2: Vast Design Architecture and Application Data Available for Harvest

FPGAs have a large device and design data farm available for harvest.

Figure 1: **Large Device and Design Data Farm**

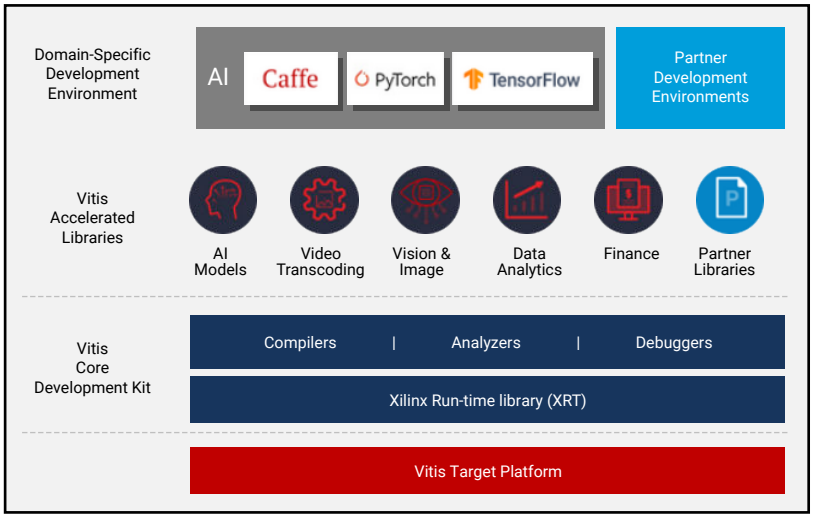


- Diverse applications (ML, emulation, automotive, DSP, 5G, aerospace/defense, high-performance computing (HPC), database, etc.)
  - Affords more robust ML models.
- Domain specific architecture (DSA) designs (convolutional neural networks (CNNs), deep neural networks (DNNs), graph analytics, etc.)
  - Presents a unique opportunity for custom DSA ML solutions at scale.

Reason 3: FPGA PD is Completely Streamlined

A single streamlined design flow from high-level description to device bitstream mapping allows full access to annotate, instrument, harvest, and optimize the PD flow.

Figure 2: **Streamlined Design Flow**



The Vivado design tools incorporate ML in various parts of its PD flow for both feature modeling and algorithm parameter optimization. Among them are a strategies recommender system, design routing difficulty predictor, routing congestion estimator, and delay estimator. In this white paper, the latter two are briefly covered.

ML-Based Routing Congestion Estimator

The increased capacity of Xilinx® devices allows for larger and higher performance designs to be mapped and realized in the logic interconnect. This is especially true for emulation, prototyping, and HPC designs in which the high logic utilization and demand for more routing resources can make routing challenging. To mitigate this, it is very important to accurately model routing congestion upstream (for yet-to-be routed nets) during the placement stage to provide the router downstream an easier placed netlist to route.

Xilinx UltraScale™ FPGAs, UltraScale+™ FPGAs, and Versal® ACAPs each have architectural blocks that control the programmable logic, block RAM, DSP, and I/O. A configurable logic block (CLB) has basic logic elements (BLEs) that contain a set of LUTs (for logic and memory implementations) and flip-flops. The BLEs are grouped into slices that share common switches to an interconnect routing grid.

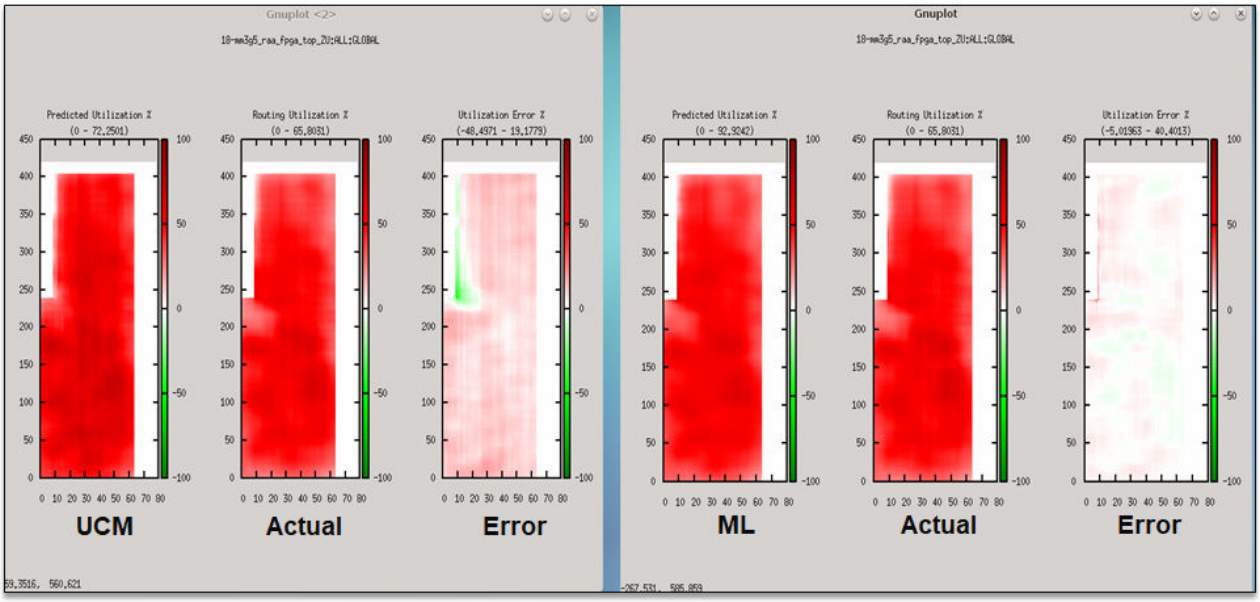
Congestion occurs when demand for the routing resources is high in certain regions. The interconnect grid routing resources are categorized by direction (north, south, east, and west) and length (short and long wires). Accordingly, a congestion map is a collection of eight maps (north/short, south/short, east/short, west/short, north/long, south/long, east/long, and west/long). When modeling congestion, it is often useful to group the short and long resources together into overall global routing resources. In this context, there are an additional four maps to model (north/global, south/global, east/global, and west/global). Short congestion is typically caused by tightly clustered cell placements potentially leading to routability issues. Long congestion is typically caused by lengthy spread-out or feedthrough nets potentially leading to timing closure and routability issues.

Along with timing and wire length (WL) metrics, the Vivado design tool's placement flow iteratively queries a routing congestion estimation oracle to identify potential local areas of routing congestion hot spots. Congestion mitigation techniques are deployed to alleviate the demand of routing resources in these areas. These include local cell inflation, target placeable area capacity reduction, logic remapping, control set optimization, macro placement optimization, and targeted net wire length reduction. During placement, the nets are not routed yet. There are up to 12 congestion map estimates per each query. The congestion estimator must be not only be highly accurate when compared to a congestion map generated by the actual router, but also fast to compute because it is queried numerous times. Aside from using a costly global routing estimator, there are various fast heuristic approaches to estimate congestion (e.g., local rent exponent, net cuts per region, overlapping net bounding box distribution, image blending, single-pass PathFinder, probabilistic rectilinear minimal Steiner tree distributions, and rectangular uniform wire density) [REF 2, REF 3, REF 4, and REF 5]. The disadvantages with these approaches are either high run time or poor accuracy. What complicates the congestion problem

is the type of routing resource the router uses to route a net, timing criticality, net fanout, routing demand due to macro placement, inter super long region (SLR) logic connectivity, etc. Maarouf et al. [REF 6], demonstrates the clear advantages of ML-based congestion estimation techniques in significantly improving accuracy over prior approaches. This white paper is partly influenced by Maarouf et al.

The accuracy of the estimated congestion maps is measured versus the actual congestion map calculated from the routing resources used by the Vivado initial router. The following figure illustrates the baseline congestion map estimator for the north short direction. The figure is a visual representation of the error map for the ML congestion estimator (CE) versus the baseline congestion estimator. Ideally, the error map should be blank.

Figure 3: **Error Map for ML Congestion Estimator vs. Baseline Congestion Estimator**

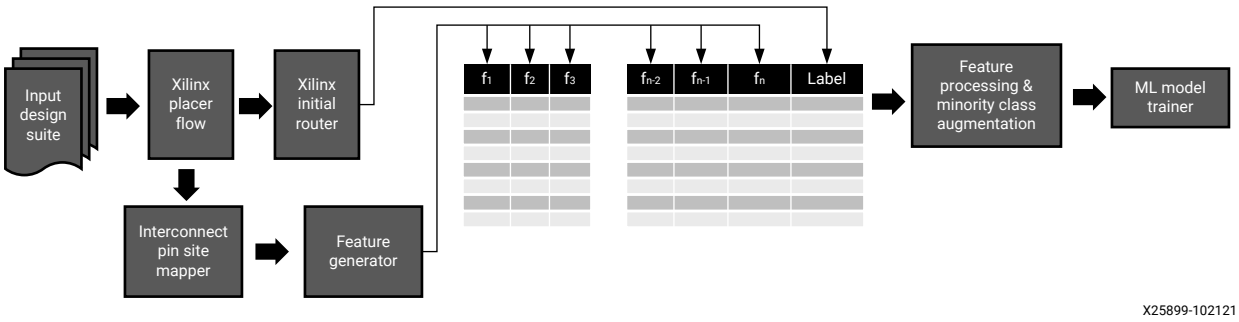


X25502-102521

The interconnect grid is a pixelated image per direction (N, S, E, and W) and per interconnect resource type (short, long, and global). The Vivado ML edition incorporates an ML framework for accurately and efficiently estimating congestion and alleviation during placement (see the following figure). Compared with the baseline approach, it demonstrates significant improvement in the congestion map correlation accuracy and reduces the router run time for hard to route suites of benchmark designs by about 10% in geometric mean and as much as 60% across multiple generations of device families.

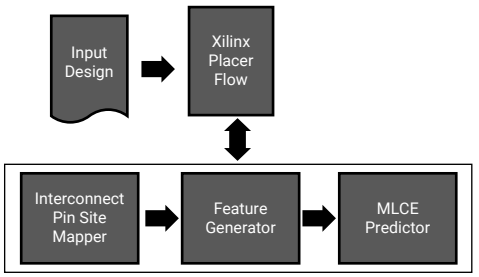
The following figure shows the MLCE training flow.

Figure 4: **MLCE Training Flow**



The following figure shows the MLCE inference flow.

Figure 5: **MLCE Inference Flow**



Feature Extraction

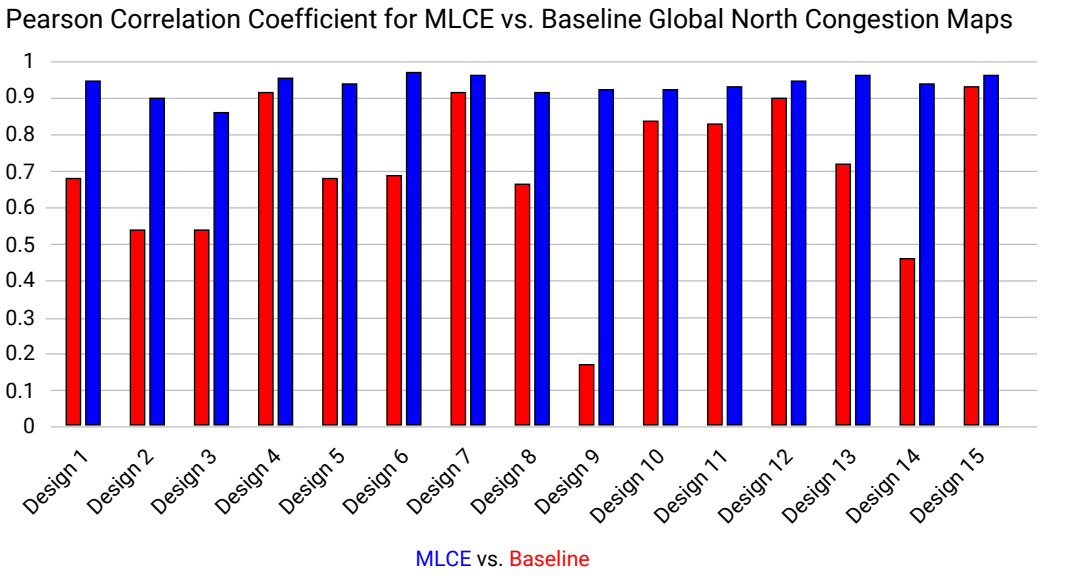
Routing congestion estimation is similar to semantic segmentation, in which given an encoded input image, an overlay is generated that predicts congestion hotspots over the image. Viewed from this perspective, a number of supervised ML (e.g., traditional regression or fully convolutional network) approaches can be adapted to estimate congestion by mapping netlist and interconnect grid features onto pixelated images and regressing on them given a target image label. For instance, Alawieh et al. [REF 9] use a conditional generative adversarial network (CGAN) approach that converts input images to labeled images to model congestion maps and demonstrates significant accuracy improvements in the Xilinx UltraScale 2016 ISPD FPGA benchmark suite. Maarouf et al. [REF 6] tabulate the features and use a random forest and MLP regressors, which demonstrates marked accuracy improvement. Zhou et al. [REF 7] encode netlist features onto interleaved RGB image channels and develop a generative adversarial network (GAN) framework where the generator is a fully convolutional encoder/decoder network, and the discriminator is a CNN that demonstrates impressive congestion map accuracy on ASIC benchmark suites. In the Vivado ML edition, each feature matrix has the same dimension as the interconnect routing grid. Features are extracted from the netlist or the interconnect tile grid. As in Maarouf et al. [REF 6], a smoothed version of the features is used to capture the impact of neighboring congested regions. The training framework for the ML-based congestion estimator is depicted in Figure 4. The features are extracted from hundreds of design suites for each device family. These features include:

- Routing resource capacities.

- Aggregate net routing resource demands.
  - Based on each net's bounding box distribution covering interconnect grid bins.
  - Based on each net's rectilinear Steiner minimal tree distribution covering interconnect grid bins.
- The number of driver and load typed pins per interconnect grid bin is used to model demand for local routing resources.
- Interconnect tile bin is a boundary tile.
- Timing criticality map based on slack distribution of driver to pin pairs.
- Cell area distribution.
- Rectilinear minimal Steiner tree route distribution.

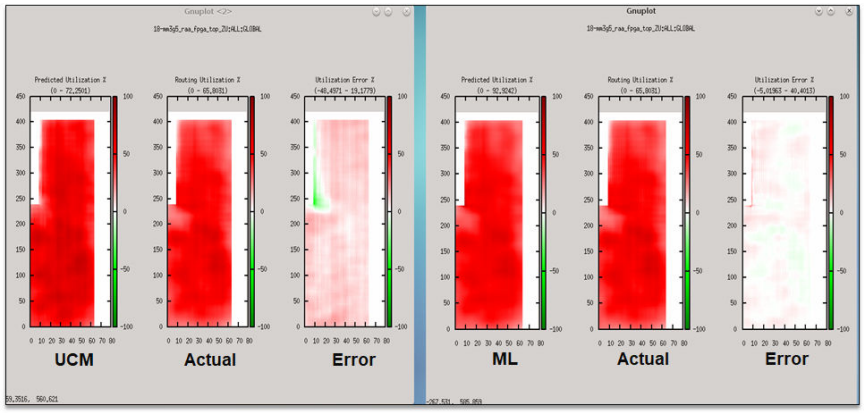
These features are blurred with convolution filters to form the final feature data set. The number and size of convolution kernels depends on whether short or long congestion is being modeled. For short congestion, smaller sized kernels are used because short congestion is a local phenomenon. For long congestion, both small and large sized kernels are used because long net routing is impacted by both local and global routing demands. During global placement, a site mapper kernel is used to infer the closest locations of logic cells to an interconnect tile for feature extraction purposes. For the training label, an actual congestion map is used over the interconnect tile grid based on a routed detail-placed design. The router is run in the shortest path mode. The congestion map is smoothed by a 4 x 4 uniform convolution filter. The following figure illustrates the accuracy gain in terms of the Pearson Correlation Coefficient for global congestion maps predicted on a suite of UltraScale+ device designs. It compares the accuracy of the ML-based and baseline congestion estimation map for an UltraScale+ device versus a congestion map calculated by the Vivado design tool router. A value of 1 means perfect correlation.

Figure 6: **Pearson Correlation Coefficient Comparing Accuracy of ML-based and Baseline Congestion Estimation Map**



The following figure illustrates the error reduction when compared with the congestion map calculated from the initially routed design. Ideally, the error map should be blank.

Figure 7: **Visual Representation of the Error Map for MLCE vs. Baseline Congestion Estimator**



X25914-102521

The following table shows the reduction in the router expansion count in four benchmark design suites when deploying the ML-based congestion estimator in the Vivado ML edition.

Table 1: **Geometric Mean of Router Expansion Count Reduction**

Benchmark Design Suite	Router Expansion Count Reduction	
	Long Pole	Overall
VU440 emulation	19%	12%
Customer default	9%	1%
Customer explore	10%	3%
VU19P prototyping	12%	12%

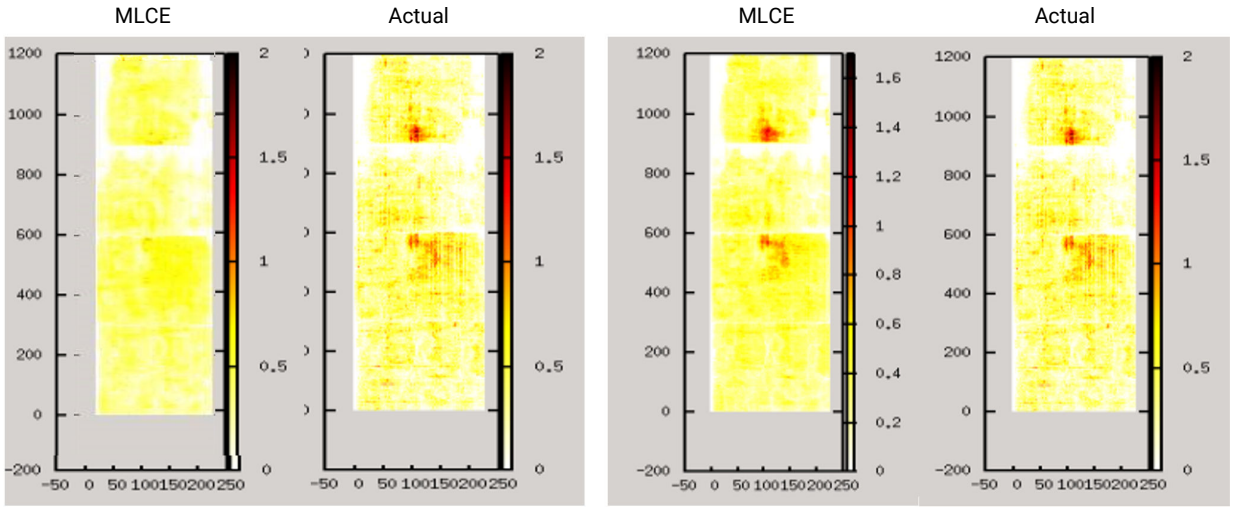
The congested regions in the die areas tend to be in the minority class based on direction, routing resource type, and design utilization. The following table shows an example of the resource utilization profile for a particular UltraScale+ device global/west sampled design data population. In this case, only 56.1K out of about 2.4M data samples show congestion (i.e., <0.05%). Consequently, it is useful to augment the sampled population of the minority class during training.

Table 2: **Resource Utilization Profile Distribution for Sampled Global/West Data**

Resource Utilization (%)	Number of Samples
0.0 – 0.5	18.2M
0.5 – 1.0	2.2M
1.0 – 1.5	56.1K
>1.5	115

The following figure shows the improvement in the congestion map accuracy before and after applying minority class augmentation for a global west congestion estimate on an UltraScale+ device design.

Figure 8: **Congestion Map Accuracy Improvement**



a. Before data augmentation

b. After data augmentation

X25544-070821

ML-Based Delay Estimator

During the PD flow, a delay estimator is required by the static timing analyzer (STA) to predict driver pin-to-load pin delays for yet-to-be routed nets. This requirement occurs during various stages such as logic synthesis, floor planning, global placement, detail placement, and timing optimization before routing is invoked. It is also used during the initial phases of the router to identify timing critical nets (or subnets) to steer the router away from detouring them.

Absent routing, these delays are estimated based on the shortest path routes. Clearly, the fidelity of these estimates greatly impacts timing closure downstream. Estimating these delays accurately is a complex task for many reasons, including:

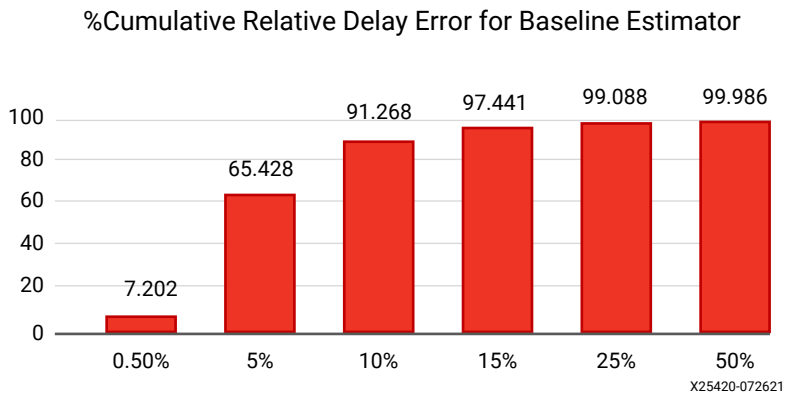
- Nature of its highly nonlinear behavior.
- Where in the fabric the respective nets reside.
- Type and number of interconnect tiles crossings among them.
- Preferred routing sites.
- Proximity to inter-die boundaries.
- Diversity of pin site types in an interconnect tile.
- Net fanout.
- Routing demand.

Consequently, developing delay estimation models is a tedious, error prone, time-consuming, and highly empirical process that needs to be done for every device family iteration. Adding to this complex process is the vast number of available interconnect pin sites that need to be modeled over multiple speed/corner variations per device family.

To understand the complexity of delay estimation, consider a device with an interconnect grid of dimensions 400 x 600 tiles, where each interconnect tile has 128 input pin sites and 64 output pin sites. To estimate the delay for short nets whose driver-to-load distance is less than 10 interconnect tiles away, the number of samples that need to be generated by sweeping the window across the device is roughly 10 x 10 x 390 x 590 x 64 x 128 x 4 = 753,991,680,000 samples. This adds to the challenge of empirically derived models.

The following figure depicts a typical cumulative error profile for an empirically derived delay estimator model for an UltraScale+ device. It shows that only about 65% of the driver-to-load connections have a relative delay error less than 5%. This error profile is undesirable because it can negatively impact timing closure. For instance, only 68.428% of driver-to-sink delays have a relative error less than 5%. Ideally, 100% of these relative errors should be less than 5%.

Figure 9: **Cumulative Error Distribution for Baseline Delay Estimator**



To address the delay estimation challenge, an automated ML-based delay estimation modeling framework has been developed. ML offers several key advantages:

- Can incorporate many features (e.g., crossing tile types, capturing the complexity of the nonlinear behavior of the delay characteristics).
- Affords increased model accuracy with larger more diverse data samples.
- Automates the modeling process across different manufacturing corners for each new or generation of device families and their respective instances.

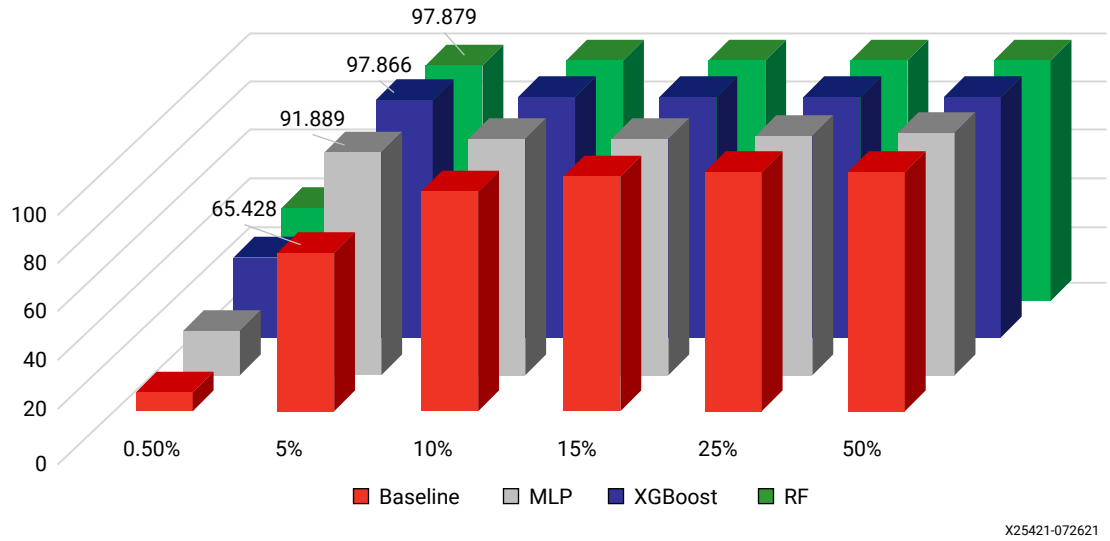
Several ML models were regressed based on a supervised learning approach. For the training framework, tens of millions of driver-to-load pairs were generated across the interconnect grid from a diverse population of interconnect sites and distances. As is typical, 80% of these samples were used for training and 20% for validation. The features included various aspects of the interconnect grid traversed by the driver to load connections such as driver and load pin source IDs, distance, number of interconnect tile grid crossing types. For the label, the calculated delay based on the routed shortest path was used.

Table 3: **Cumulative Relative Errors for XGBoost Delay Model vs. Empirical Baseline Model**

Error	Baseline Delay Estimator	MLDE
0%	7.47	36.04
2%	28.34	84.90
5%	67.90	98.15
10%	94.29	99.82
15%	98.82	99.96
20%	99.65	99.99
25%	99.65	100.00
50%	99.55	100.00

The following figure shows the cumulative error chart for various ML regression models (a 4-layer multi-perceptron neural net, random forest, and XGBoost). It shows the comparison of the cumulative relative errors for three different ML-based delay models versus the baseline empirical model for an UltraScale+ device. Clearly, there is a marked gain in accuracy against the baseline delay model. Similarly, Table 3 shows the cumulative errors for the ML-based model versus the baseline model for a Versal device. Notice in particular, the improved accuracy for pin pairs with errors less than or equal to 5%. The percentage of sampled data with less than 5% error increases to 97.866.

Figure 10: **Cumulative Relative Error for ML Delay Models vs. Baseline**

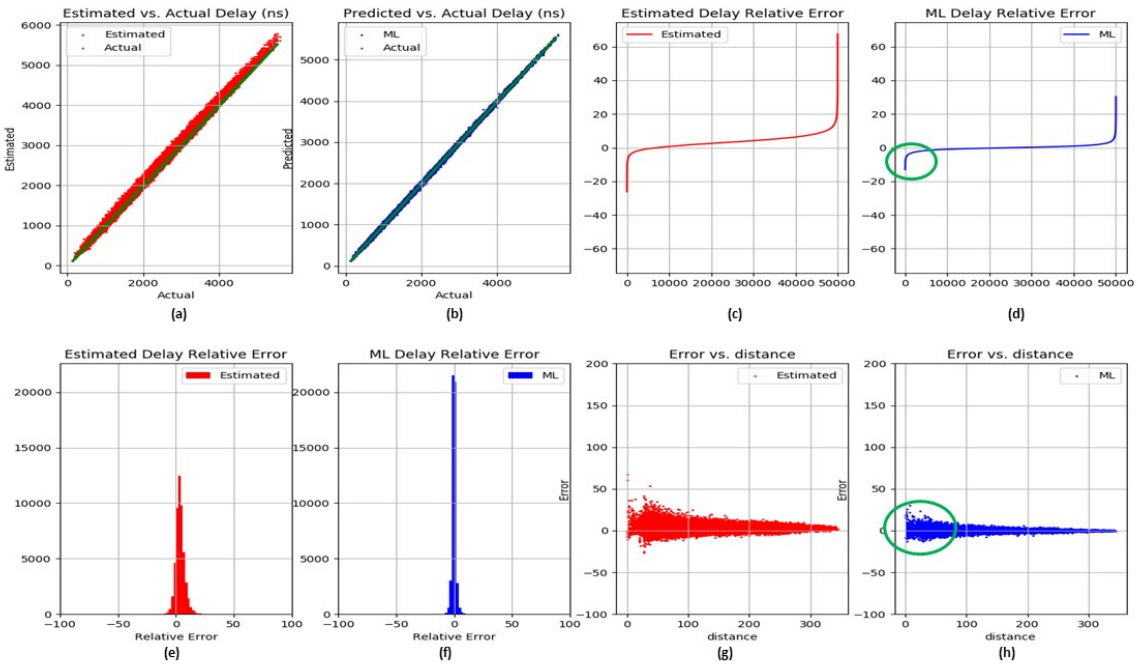


XGBoost was chosen because it is highly accurate and has a mature C++ API for inference. Aside from training, a main challenge for deploying a ML-based delay estimator is the inference run time. This is a particularly strict requirement because the timer queries delay estimates repeatedly during the various parts of the PD flow. Another complicating issue is that the delay is typically estimated by iterating over each net rather than all together, which slows the cache. Consequently, addressing the inference run time was more challenging than fine tuning the modeling accuracy. To solve the run time issue, aside from the obvious tweaking of ML model parameters (e.g., number of tree estimators, maximum tree depth, etc.), a fast C++ inference

framework based on three optimizations was developed. First, an in-memory compiled C++ ML model with an efficient per net query API was developed. Second, tile types with similar delay characteristics were aliased to reduce the number of crossing type features. Third, pin site locations were cached for fast look up of distance metrics. The latter is an optimization that takes advantage of the fact that the interconnect logic layout in an FPGA is known (a feature not always common in ASICs). These optimizations helped reduce inference run time by several orders of magnitude. This enabled the deployment of highly accurate ML-based delay models in the Vivado tools with on-par run times close to the baseline models.

The following figure shows comparative error plots for ML XGBoost-based delay models versus the baseline model for a Versal ACAP. Plot a and b show the scatter plot comparison. Plot c and d show sorted relative error comparison. Notice how the ML model is much more accurate and has much fewer number of outliers. Plot e and f show a comparison of the error histograms. Plot g and h show the relative error as a function of the Manhattan distance between the driver and the sink pairs.

Figure 11: **Comparative Error Plots for the ML-Based Delay Estimate Model vs. Baseline Model**



X25545-070821

Conclusion

In this white paper, two instances are described in which an ML framework has been deployed in the Vivado ML edition to improve model accuracy, namely for delay and routing congestion estimation. Aside from the clear gains in automating the model generation process, much higher accuracy, and improved quality of results, some insights are shared in this conclusion.

In addition to the usual ML model parameters' tuning and data augmentation required during the training process, ML inference is a challenge because inference run time is key for online applications such as the Vivado ML edition. Much work went into developing custom-optimized C++ (rather than Python) API's and streamlining feature extraction run times. It is also important to develop a ML model lifecycle management plan, such as when to trigger model updates and how to devise a versioning and model compatibility plan. Furthermore, ML modeling and algorithm parameter tuning are synergistic. Better modeling through ML does not automatically guarantee better quality of results. It often needs to be coupled with retuning (or optimizing) the respective algorithms for each deployed application. Finally, there is promising work in progress for the development of ML frameworks for solving optimization problems. Adaptations of this work will greatly improve the physical design flow.

References

These documents provide supplemental material useful with this white paper:

1. Guyue Huang, Jingbo Hu, Yifan He, Jialong Liu, Mingyuan Ma, Zhaoyang Shen, Juejian Wu, Yuanfan Xu, Hengrui Zhang, Kai, Zhong, Xuefei Ning, Yuzhe Ma, Haoyu Yang, Bei Yu, Huazhong Yang, Yu Wang, *Machine Learning for Electronic Design Automation: A Survey*, arXiv:2102.03357v2, 2021.
2. D. Yeager, D. Chiu, and G. Lemieux. *Congestion Estimation and Localization in FPGAs: A Visual Tool for Interconnect Prediction*. In International Workshop on System Level Interconnect Prediction, pages 33–40. ACM, 2007.
3. J. Swartz, V. Betz, and J. Rose. *A Fast Routability-driven Router for FPGAs*. In International Symposium on FPGAs, pages 140–149. ACM, 1998.
4. C. Pui, G. Chen, Y. Ma, E. Young, and B. Yu. *Clock-Aware UltraScale FPGA Placement with Machine Learning Routability Prediction*. In International Conference on Computer Aided Design, pages 929–936. ACM, 2017.
5. P. Spindler and F. M. Johannes, *Fast and Accurate Routing Demand Estimation for Efficient Routability-driven Placement*, 2007 Design, Automation & Test in Europe Conference & Exhibition, 2007, pp. 1-6, doi: 10.1109/DATE.2007.364463.
6. D. Maarouf et al., *Machine-Learning Based Congestion Estimation for Modern FPGAs*, 2018 28th International Conference on Field Programmable Logic and Applications (FPL), 2018, pp. 427-4277, doi: 10.1109/FPL.2018.00079.
7. Z. Zhou, Z. Zhu, J. Chen, Y. Ma, B. Yu, T. Ho, G. Lemieux, and A. Ivanov, *Congestion-aware global routing using deep convolutional generative adversarial networks*, in 2019 ACM/IEEE 1st Workshop on Machine Learning for CAD (MLCAD), 2019, pp. 1–6.
8. J. Masci, U. Meier, D. Cires¸an, and J. Schmidhuber, *Stacked convolutional auto-encoders for hierarchical feature extraction*, in Artificial Neural Networks - Volume Part I, 2011, pp. 52–59.
9. 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 2020 25th Asia and South Pacific.

Revision History

The following table shows the revision history for this document.

Section	Revision Summary
10/26/2021 Version 1.0	
Initial release.	N/A

Please Read: Important Legal Notices