Machine-Learning Based Congestion Estimation for Modern FPGAs

D. Maarouf, A. Alhyari, Z. Abuowaimer, T. Martin, A. Gunter, G. Grewal, S. Areibi, A. Vannelli
School of Engineering/School of Computer Science, University of Guelph
Guelph, Ontario, Canada
{dmaarouf,aalhyari,abuowaiz,tmarti14,agunter,ggrewal,sareibi}@uoguelph.ca, Tony.vannelli@usask.ca

***Abstract*—Avoiding congestion for routing resources has become one of the most important placement objectives. In this paper, we present a machine-learning model for accurately and efficiently estimating congestion during FPGA placement. Compared with the state-of-the-art machine-learning congestion-estimation model, our results show a 25% improvement in prediction accuracy. This makes our model competitive with congestion estimates produced using a global router. However, our model runs, on average, 291x faster than the global router.**

***Keywords*-FPGA; Congestion; Machine Learning**

I. Introduction

Placement tools that seek only to optimize wirelength and/or timing without also taking into account the (fixed) routing resources present on the FPGA device may produce placements that exhibit unacceptably low performance, or placements which are unroutable [2]. An effective way to improve the routability of any design targeting an FPGA is to incorporate *congestion awareness* into the placement flow. By first identifying local regions of congestion, cell inflation can be performed to ensure that the routing resources in these regions are no longer exceeded. However, the ultimate success of the previous approach greatly depends on the congestion estimation technique being both *fast* and *accurate*. The former is necessary as estimating congestion may have to be performed numerous times while optimizing a placement, especially if the circuit being placed is highly congested. The latter is necessary because a large gap between estimated congestion and the actual performance of the global and detailed routers may result in degraded performance or an unroutable placement. Many techniques of various kinds have been proposed for estimating FPGA congestion. The most computationally efficient techniques are based on using fast-to-compute heuristics [16]. However, the resulting congestion estimates are often far away from the actual congestion encountered by the detailed router. Consequently, others have sought to obtain accurate congestion estimates by running a global router [11] during placement. This approach results in more accurate congestion estimates, but at the cost of runtime. In fact, the cost of running the global router is so prohibitive, often the router can only be run a few times during placement, potentially limiting its overall effectiveness.

Recently, research has been directed toward using linear-regression to model and predict routing congestion [10]. When tested using the 12 ISPD 2016 Routing-Driven FPGA Placement contest benchmarks [15], the regression model achieves an accuracy of 90% compared with the congestion estimates produced by the Vivado Design Suite. However, the experimental results that we present in this paper show that when used to predict the actual congestion encountered by the Vivado detailed router, the accuracy of the previous model [10] drops to 60%. This result serves to highlight the fact that when seeking to solve a complex problem, like estimating congestion, features must be carefully prepared to characterize the underlying problem. When done correctly, such features often allow for the use of a simple model that can be computed quickly, which works well when presented with new input data, and that is not overly sensitive to tuning parameters.

Motivated by the work in [10], we present an efficient machine-learning model for accurately predicting actual congestion. Our model includes one feature from [10], but introduces three new features to capture the subtleties of the underlying congestion problem. When tested with the 12 ISPD 2016 contest benchmarks, along with 360 benchmarks provided by Xilinx Inc., our model achieves a 25% increase in accuracy compared with [10] when predicting actual congestion. It also runs (on average) 291x faster than the global router employed in [1], with no loss in accuracy.

The remainder of this paper is organized as follows. In Section II a detailed description of the heterogeneous UltraScale FPGA architecture used in this work is presented along with related work published. The main methodology and framework proposed in this paper are introduced in Section III. The experimental setup and detailed results and analysis are presented in Section IV. Section V presents our conclusions and future work.

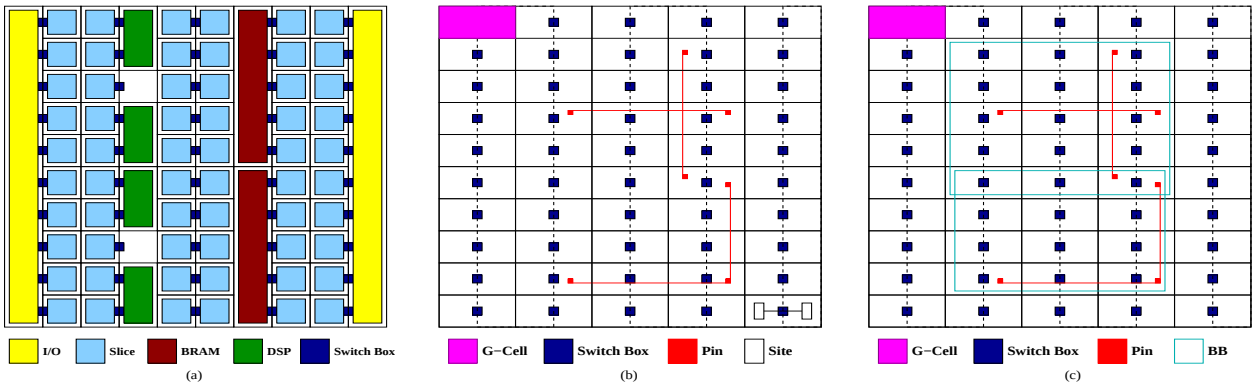


Fig. 1: Xilinx UltraScale Architecture

II. Background and Previous Work

Our work in this paper is based on the UltraScale FPGA architecture introduced recently by Xilinx [14]. The UltraScale FPGA, as shown in Fig. 1(a), consists of heterogeneous programmable logic blocks such as Slices, Random Access Memory Blocks (i.e., BRAMs), Digital-Signal Processing blocks (i.e., DSPs) and I/O. A Slice consists of a single Configurable Logic Block (CLB) which can contain up to eight Basic Logic Elements (BLE). Each BLE can contain up to two Look Up Tables (LUTs) and two Flip Flops (FFs). As well, there are prefabricated routing segments of different lengths in both horizontal and vertical directions. LUTs and FFs when grouped together within a Slice share a single switch for routing. Fig. 1(b) shows in more detail the routing architecture of the FPGA and demonstrates how the Switch Boxes can provide both intra- and inter-slice connectivity. A routing region is uniformly formed by vertical and horizontal evenly spaced grid lines that create a uniform grid over the FPGA fabric. The latter is composed of unit global cells (i.e., gcells). Each gcell has a number of vertical and horizontal routing resources that can be used in congestion estimation. Fig. 1(c) shows the Bounding Box (BB) or Half Perimeter WireLength (HPWL) used to estimate the length of a net. In general, as the number of overlapping nets inside a region increases, the Switch Box utilization also increases, thus affecting congestion in the region.

A. Related Work

The fixed resources on an FPGA fabric provides a unique challenge to routing, and so many congestion estimation techniques have been developed specifically for FPGAs. The authors in [17] proposed a congestion driven placement approach which counts the number of overlapping bounding boxes of all nets in a region, and assigns that as the congestion value of that region (known as BB cost). However, this technique tends to neglect the effect of larger bounding boxes overlapping smaller bounding boxes, thus reducing the accuracy of the congestion estimate. [16] introduced the Wire Length Per Area (WLPA) which improves upon the BB cost. Unlike BB cost [17], which simply counts the number of overlapping bounding boxes within a region, WLPA takes into account the sizes of each bounding box covering a region. Specifically, WLPA estimates the amount of wire that a net will use, then distributes that amount over the gcell(s) within its bounding-box. Another important measure proposed by [16] is the Net Cuts per Region (NCPR) which is both fast and used as a direct indicator for congestion prediction. However, this technique tends to produce discontinuous congestion regions that are not reflected in real congestion maps.

A well known congestion estimate developed specifically for FPGAs is fGREP. Presented in [6], fGREP employs a routing resource graph to provide a detailed prediction of congestion. The method is extended to fGREP2 in [7], with the goal of improving runtime. While fGREP and fGREP2 generally provide a relatively accurate, detailed estimate of congestion at the cost of a slightly worse runtime compared to some less precise estimates, the full variety and limitations of routing resources available to the FPGA are still not considered.

In [3], routability is predicted by examining all available routing resources. This paper uses a probabilistic analysis approach to estimate the number of resources required to be used by a design. This can provide a prediction of a circuit's routability before placement. However, probabilistic congestion estimation methods tend to be very slow and thus useless for large designs.

A series of estimation techniques that take the FPGA limitations of fixed channel widths and wire lengths into account are presented by Lemieux [16]. In these methods, congestion estimations are taken from several existing techniques and processed as images. An image blending method is presented, which iteratively spreads

the congested regions in the image to nearby areas. A peak saturation method is also proposed to model the fixed channel widths by expanding the most congested regions. Congestion estimation techniques such as WLPA improved using one or both of these methods. While these estimates provide useful improvements, they still do not directly consider the FPGA features they seek to model, and are tuned experimentally for each base congestion estimation technique.

Most recently, a machine learning approach based on a linear regression model was used to estimate congestion in [10]. Their model uses three features from each site based on WLPA and pin count, along with the features from surrounding sites. Once trained, the model is able to estimate the congestion *prediction* given by Vivado; however, it is less accurate as a predictor of the actual congestion found by the detailed router. There are several issues with the work in [10] that we seek to address in this paper including: (i) feature engineering, as will be explained in Section IV, (ii) the number of benchmarks used to train the machine learning model, and (iii) the accuracy of congestion estimation reported.

III. Proposed Framework

Our methodology for constructing a machine-learning model to quickly and accurately predict congestion during placement is illustrated in Fig. 2. The proposed framework consists of an *off-line* training and testing stage, and an *on-line* deployment stage. The steps to be taken in each stage are described below.

A. Step 1: Benchmarks to be used

To train and test the prediction model, we start with a large set of benchmarks that includes the 12 ISPD 2016 contest benchmarks [13], and an additional 360 benchmarks provided directly by Xilinx Inc. The latter benchmarks were created using the netlist generation tool, Gnl [5], and target a modern Xilinx UltraScale FPGA device. During synthesis, key circuit features were varied to create a rich set of benchmarks, as shown in Table I. The benchmarks range in size from 0.1 to 1.1 million gates, and produce highly congested placements, as the results in Table X will later show.

TABLE I: Range of Key Circuit Properties [5]

#LUTs	#FF	#BRAM	#DSP	#CSet	#IO	R.E
44K-518K	52K-630K	0-1035	0-620	11-2684	150-600	0.4-0.8

B. Step 2: Creating Training Data

To have a predictive model perform well on new data, it is desirable to have a large, representative set of data (i.e., records) with which to train and test the model. With this in mind, we initially create 28,478,400 records as follows: First, two state-of-the-art academic placers, [8] and Ripple [9], are used to create actual placement solutions for each of the 372 benchmarks (see Step 1). Second, in keeping with the UltraScale FPGA device having 480 rows and 85 columns, each placement solution is partitioned into a grid containing 480 x 85 equal-size regions, where each region corresponds to a gcell. From this point forward, each gcell represents a single record that can be used either for training or testing purposes.

C. Step 3: Feature Extraction

As illustrated in Fig. 2, each record contains four features and a single label. The label is a continuous output variable in the range [0,1] that corresponds to the actual congestion in the corresponding gcell. The features are continuous input variables which are designed to estimate how the placement solution will impact the routing stage. The features are generic and easy to compute, and when combined together can accurately predict the actual congestion value of a gcell. Each feature is described in detail below:

1) **WireLength Per Area (WLPA)**: Estimates the routing demand of each net by estimating the wirelength of the net and then distributing the resulting demand over the gcells covered by the bounding box of the nets' pins. The router tends to route nets within the nets bounding box, therefore WLPA helps to identify gcells where many nets will contend for routing resources. The WLPA of a placement is calculated as:

$$f_1 = \sum_{n \in N_i} \frac{w_n . HPWL_n}{\#gcell_n} \tag{1}$$

where $HPWL_n$ is the half perimeter wirelength of net n, w_n is a correction factor used in VPR to improve the accuracy of HPWL on high fan-out nets, N_i is the set of nets whose bounding box overlap a gcell, and $\#gcell_n$ is the number of gcells covered by net n.

2) **Pin Count**: The number of pins within a gcell is used to model the *local* congestion of a gcell. Other features such as WLPA are designed to estimate the routing demand on gcells due to inter-switch routing, but do not consider intra-switch routing demand. The internal routing flexibility of a switch is limited, therefore, higher demand on local routing resources results not only in internal congestion for the gcell, but also further degrades the switches flexibility and increases the likelihood that the routing trees of local connections will

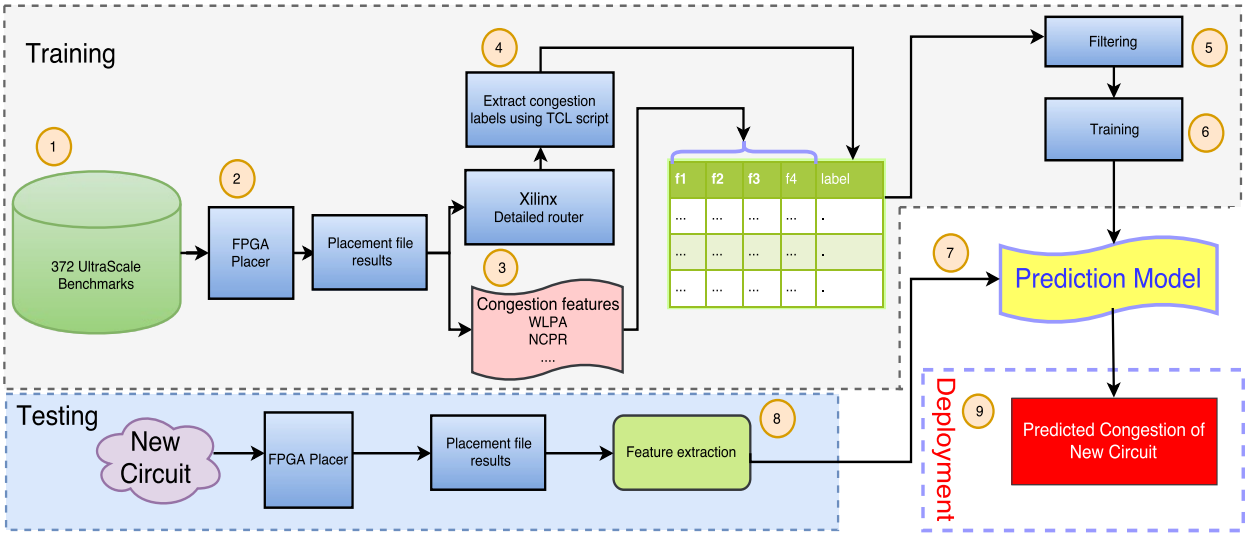


Fig. 2: Machine Learning Framework: Training and Deployment

overflow into surrounding gcells contributing to global congestion. This feature is calculated as:

$$f_2 = \sum_{n \in N_i} \#pins_{n,gcell_i} \quad (2)$$

where N_i is the set of nets with at least one pin in the gcell, and $\#pins_{n,gcell_i}$ is the number of pins of net n residing in $gcell_i$.

3) **Nets Cut Per Region**: The absolute number of nets cut in a physical region spanning multiple gcells is also used to model routing demand. A large number of cut nets indicates that many nets must enter or leave the region during routing, which indicates that there is high demand on the inter-region global routing resources. Using larger window sizes allows a more global view of the demand on routing resources to be captured. Two window sizes are considered: 5×5 and 9×9.

$$f_3 = \mid W_{5x5} \mid \quad (3)$$

$$f_4 = \mid W_{9x9} \mid \quad (4)$$

W_{jxj} is the set of connected nets within a window of size $j \times j$, where each net has at least one pin inside the physical region specified by the window and at least one pin outside of the region.

It is important to note that all of the previous features require little effort to compute. This is important, since congestion prediction may need to be performed frequently during placement once the model is deployed.

D. Step 4: Label Extraction

To complete each record, the label or actual congestion estimate for each gcell is computed. The actual congestion is determined by first using the Vivado Router to route each of the placements produced in step 2 for the original 372 benchmarks. Then, a *Global Routing Resource (GRR)* graph is constructed. The first step in constructing the GRR graph for the UltraScale FPGA architecture is to use Tincr [12] to generate an XDLRC file. The XDLRC file is then parsed, and a compact representation of the GRR graph is built. Using the GRR graph, the interconnect demands on each channel for each switch box (within each gcell) are determined. The actual congestion value of a switch box is calculated as the sum of the interconnect demands of each outgoing channel divided by the sum of the capacities of each channel. The congestion value of the switch box is used as the label for the record.

E. Step 5: Filtering

After step 4, it is possible that some of the records will correspond to unused regions on the FPGA device. (This will be the case if the pin count or the congestion in a gcell is zero.) In the current dataset, 7,125,418 such records were found. These were filtered and removed, leaving a total of 21,352,982 valid records for training and testing.

F. Step 6: Training and Testing

To avoid bias when creating and evaluating a model, the same records should not be used to both create and test a model. Rather, the records should be partitioned into two independent sets: a larger set, which are used to train the model, and another smaller set, which are used to test (or assess) the performance of the model. We randomly select 70% of the records from step 5 to

appear in the training set, while the remaining 30% are used for testing and analysis purposes.

G. Step 7: Machine-Learning Models

The performance of different machine-learning models for performing prediction strongly depends on the structure and size of the data used to build the models. Therefore, the correct choice of model often remains unclear unless several models are tested and compared. In this paper, we implement and compare several prediction models, and highlight their advantages and disadvantages:

1) *Linear Regression* is a technique used to model the relationship between several independent variables and a single dependent variable. This technique is fast and useful when the relationship to be modeled is not complex.
2) *K-Nearest Neighbor* (KNN) is a non-parametric technique that utilizes the responses of the k-nearest neighbors in the training data to form a prediction.
3) *Artificial Neural Networks (ANNs)* are effective at modeling non-linear relationships, and are very flexible in learning almost any kind of feature variable relationships. However, they can be quite challenging and computationally intensive to train, since they require careful hyper-parameter tuning.
4) *Random Decision Forests* are also effective at learning highly complex, non-linear relationships, and are easy to understand and interpret. However, they can be prone to major over-fitting.

H. Step 8: Testing and Evaluation Metrics

The performance of each of the models created in step 7 is now determined using the testing data from Step 6. As there is no one best metric for determining predictive accuracy, we employ the following three popular measures where i and j are used as indices to specify the row and column location of a gcell in the $N \times M$ FPGA array, $y_{i,j}$ represents the actual congestion value extracted from the router for a gcell located in row i column j, $\hat{y}_{i,j}$ is the estimated congestion value for a gcell located in row i and column j, and y_{max} is a normalizing factor equal to the maximum real congestion value:

1) Mean Absolute Error (MAE):

$$MAE = \frac{1}{N} \sum_{i=1}^{N} \sum_{j=1}^{M} \mid y_{i,j} - \hat{y}_{i,j} \mid \qquad (5)$$

This measure is the average size of the absolute relative error.

2) The Root Mean Square Error (RMSE):

$$RMSE = \sqrt{\frac{1}{N \times M} \sum_{i=1}^{N} \sum_{j=1}^{M} \left(y_{i,j} - \hat{y}_{i,j} \right)^2} \qquad (6)$$

This measure gives greater weight (or emphasis) to larger errors.

3) The R-squared metric (R^2):

$$R^2 = 1 - \frac{\sum_{i=1}^{N} \sum_{j=1}^{M} (y_{i,j} - \hat{y}_{i,j})^2}{\sum_{i=1}^{N} \sum_{j=1}^{M} (y_{i,j} - \bar{y})^2} \qquad (7)$$

This gives a measure of how well the model replicates the observed outcomes.

We also employ the following two common metrics for directly comparing congestion estimates:

1) The Average Absolute Normalized Error (A.N.N.E):

$$a.a.n.e = \frac{1}{N \times M} \sum_{i=1}^{N} \sum_{j=1}^{M} \frac{\mid y_{i,j} - \hat{y}_{i,j} \mid}{y_{max}} \qquad (8)$$

Unlike $RMSE$, this measure is not biased towards larger errors.

2) The Sum of Absolute Error (SAD):

$$sad = \sum_{i=1}^{N} \sum_{j=1}^{M} \mid y_{i,j} - \hat{y}_{i,j} \mid \qquad (9)$$

This measure can be used to measure the disparity between an estimated congestion and a golden reference (e.g. produced by a router).

I. Step 9: Deployment

As a final step, the prediction model is integrated into a state-of-the-art placement flow [8], and the overall performance of the placer is analyzed.

IV. Results and Discussion

Experiments were carried out using a Linux machine running on a Xeon 3.2GHz processor. The Scikit-Learn [4] Python module was used to implement the various regression models. All routing was performed by Vivado (version 15) with a patch applied to make it compatible with the modified bookshelf benchmark format used in the ISPD 2016 FPGA placement contest.

A. Results Reported in [10]

In [10], the authors introduce a regression model based on the following three features extracted from each gcell:

1) x_1 is the WLPA given in Eqn. 1 of this paper.
2) x_2 = $\sum_{m \in N_i} \#pins\ \ of\ \ net\ \ m.$
3) x_3 = $\sum_{m \in N_i} \frac{\#pins_{m,gcell_i}}{\#pins\ of\ net\ m} \times \frac{w_m . HPWL_m}{\#gcell_m}$.

Table II shows the impact of the three selected features in [10]. It is clear from Table II that feature x_3 has little or no affect on training the linear regression model, and that feature x_2 has only a small contribution on its own. When both x_1 and x_2 are combined, the R^2 measure increases slightly from 57.5% to 60.2%. To encourage

TABLE II: Comparison of Features from [10]

Regression Models	Features			Accuracy Measures		
	x_1	x_2	x_3	MAE	RMSE	R^2
HK_{M1}	*			0.0839	0.1044	57.57%
HK_{M2}		*		0.1124	0.1412	22.35%
HK_{M3}			*	0.1280	0.1601	0.13%
HK_{M4}	*	*		0.0805	0.1011	60.25%
HK_{M5}	*		*	0.0839	0.1043	57.66%
HK_{M6}		*	*	0.1116	0.1397	24.02%
HK_{M7}	*	*	*	0.0801	0.1005	60.39%

the model to capture global congestion information, the authors in [10] replicate the three features from eight neighboring sites and add them to each gcell. A total of 27 features per gcell are subsequently used to train and test the model.

Our first step was to reproduce the results obtained by [10], as shown in Table III. We used the same three features to recreate the Local Linear (LL) model, and the 27 features to recreate the Global Linear (GL) model. The 372 benchmarks described in Section III-A were placed, features were extracted, and both LL and GL models were trained and tested. The results in Table III

TABLE III: LL and GL Results for [10]

Model	Vivado Estimated		Vivado Detailed Router	
	R^2	MAE	R^2	MAE
LL	95.1%	4.49%	60.4%	8.01%
GL	95.5%	4.22%	61.7%	7.90%

clearly shows that both models achieve an accuracy slightly greater than 95%, but only when compared with the congestion estimates produced by the Vivado Design Suite. When compared to the actual (i.e., true) congestion obtained following routing, the accuracy of the models drops closer to 60%. These results show that the features used in [10] do not adequately characterize the underlying congestion encountered by the router. In fact, they seem to suggest that the Vivado estimate itself may be based primarily on WLPA.

B. Proposed Machine Learning Framework: Results

Next, we use the framework described in Sec. III to create a linear-regression model using our own features, with the aim of achieving a higher accuracy compared to [10] when predicting actual congestion. To assess the individual and combined impact of the four features introduced earlier in Sec. III-C, we perform a full-factorial experiment by training and testing fifteen linear-regression models (i.e., G_{M1} - G_{M15}). The results are shown in Table IV.

TABLE IV: Comparison of Features for Our Model

Regression Models	Features				Accuracy Measures		
	f_1	f_2	f_3	f_4	MAE	RMSE	R^2
G_{M1}	*				0.0839	0.1044	57.57%
G_{M2}		*			0.1218	0.1496	12.87%
G_{M3}			*		0.0709	0.0889	69.22%
G_{M4}				*	0.0681	0.0852	71.78%
G_{M5}	*	*			0.0808	0.1001	60.94%
G_{M6}	*		*		0.0547	0.0692	81.36%
G_{M7}	*			*	0.0590	0.0740	78.69%
G_{M8}		*	*		0.0683	0.0863	71.03%
G_{M9}		*		*	0.0614	0.0776	76.57%
G_{M10}			*	*	0.0649	0.0814	74.19%
G_{M11}	*	*	*		0.0530	0.0673	82.35%
G_{M12}	*	*		*	0.0541	0.0684	81.81%
G_{M13}	*		*	*	0.0537	0.0679	82.03%
G_{M14}		*	*	*	0.0600	0.0760	77.49%
G_{M15}	*	*	*	*	0.0510	0.0649	83.61%

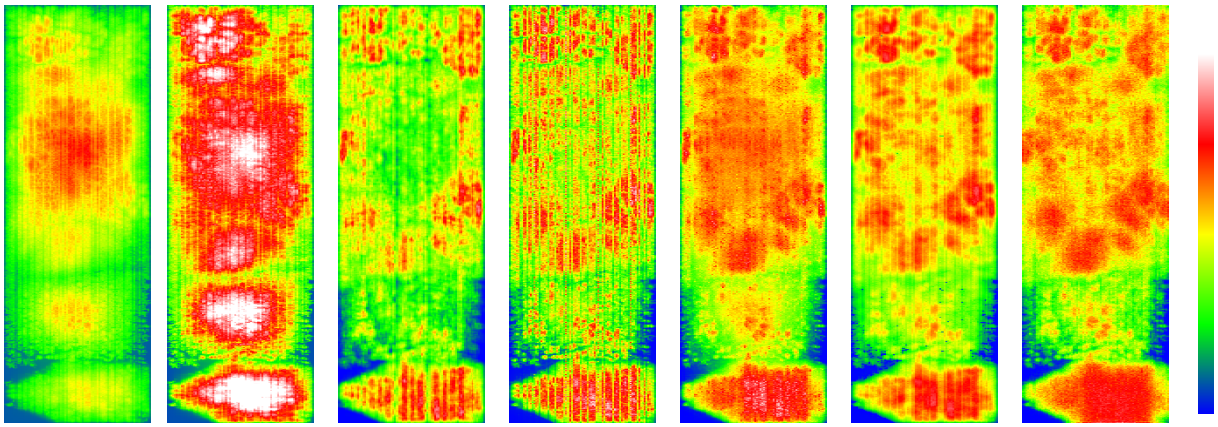
Our results show that all four features (i.e., f_1 - f_4) have predictive power ranging from 12.87% to 71.78%. The highest accuracy achieved is 83.61%, and occurs when all four features are employed (i.e., G_{M15}). In [16], Lemieux shows that the accuracy of a congestion-estimation method can be increased by performing smoothing. We explore this in our own model, by calculating the average value of feature $f3$ in the 5x5 window centered on each gcell, and using this value as an additional (smoothing) feature. Results in Table V show that including this smoothed feature improves the accuracy to 85.2%. This can be understood in light of the fact that the congestion values of adjacent gcells tend to change continuously and gradually in the real congestion map produced by the Vivado detailed router, while large, noncontinuous jumps are common in the congestion maps produced using Nets Cut Per Region (NCPR).

TABLE V: Our Features with Smoothing

Regression Models	Features					Accuracy Measures		
	f_1	f_2	f_3	f_4	Smoothing	MAE	RMSE	R^2
G_{M15}	*	*	*	*		0.0510%	0.0649%	83.61%
G_{M16}	*	*	*	*	*	0.0479%	0.0616%	85.24%

C. Regression Models: A Comparison

Several regression models are now compared in terms of accuracy and performance, as seen in Table VI (based on 70% training, 30% testing). Hyper-parameter tuning of all models was performed on all regression models. The accuracy results reported in the Table VI are uniformly high and comparable.



(a) A.A.N.E = 15.59% (b) A.A.N.E = 18.59% (c) A.A.N.E = 12.50% (d) A.A.N.E = 9.81% (e) A.A.N.E = 5.80% (f) A.A.N.E = 5.33% (g) A.A.N.E = 0.00%

Fig. 3: (a) HK_{M7}[10] (b) WLPA (c) NCPR (d) fGREP (e) mPFGR (f) G_{M16} (g) Vivado Detailed Router

TABLE VI: Regression Methods (70/30 Train/Test)

Regression Techniques	Accuracy Measures			CPU Time (s)	
	R^2	MAE	RMSE	Training	Testing
G_{M16}	85.24%	0.0479	0.0616	4.17	0.077
KNN	85.41%	0.0471	0.0612	738.23	595.24
Random For.	85.99%	0.0462	0.0599	1513.37	87.52
MLP	85.97%	0.0464	0.0600	603.63	0.53

D. Overfitting

A common issue in machine learning is overfitting which usually occurs when a model captures both the signal of interest and noise present in a dataset. When the model used for regression is complex, instead of generalizing and approximating the true model for the entire sample of data, it tends to memorize, thus fitting random noise in the specific sample and data used. To confirm that proposed model in this work does not overfit, we first use cross-validation that can detect overfit models by determining how well our model generalizes to other data sets by partitioning the data. Table VII

TABLE VII: Regression Models (10-Fold CV)

Regression Techniques	Accuracy Measures			CPU Time (s)	
	R^2	MAE	RMSE	Training	Testing
G_{M16}	85.23%	0.0479	0.0615	5.34	0.026
KNN	85.41%	0.0471	0.0612	1108.61	139.245
Random For.	85.99%	0.0462	0.0599	1640.70	47.584
MLP	86.16%	0.0461	0.0597	583.73	0.194

shows similar results to that in Table VI, but based on 10-fold cross validation. This provides evidence that the regression model used in this work is not overfitting.

The second step we take to establish that our model is not overfitting is by using regularization techniques that add a penalty for model complexity to reduce overfitting. In this experiment, we use both *Lasso Regression* and *Ridge Regression*. Table VIII clearly indicates that results obtained by the simple regression model are almost identical to those obtained by both *Lasso* and *Ridge* which confirms that the former generalizes without memorizing.

TABLE VIII: G_{M16} vs. Lasso vs. Ridge

Regression Techniques	Accuracy Measures			CPU Time (s)	
	R^2	MAE	RMSE	Training	Testing
G_{M16}	85.23%	0.0479	0.0615	5.01	0.022
Lasso Reg.	85.18%	0.0481	0.0616	5.56	0.028
Ridge Reg.	85.23%	0.0479	0.0615	1.98	0.027

E. Congestion Estimation Techniques: A Comparison

Next, we compare several congestion estimation techniques in the form of WLPA, NCPR, fGREP, a global router (mPFGR) [8], the machine-learning model in [10] (HK_{M7}), along with our machine-learning model (G_{M16}) based on linear regression. Table IX compares the previous methods based on SAD, AANE, RMSE, and R^2 for all 372 benchmarks, and shows that our model, followed by the global router give the best congestion estimates. Fig. 3 shows the individual heatmaps produced by the different methods. Although only for a single benchmark, these maps are typical of what was obtained for the other benchmarks.

TABLE IX: Congestion Estimation Techniques

Cong. Estimation Methods	Congestion Metrics		Accuracy Measures	
	SAD	a.a.n.e	RMSE	R^2
G_{M16}	2891.28	6.73%	5.93%	85.24%
mPFGR [1]	3126.27	7.34%	6.41%	-
fGREP [7]	4351.59	9.66%	8.93%	-
NCPR [16]	5254.11	11.47%	10.07%	-
WLPA [16]	6185.44	14.20%	12.30%	-
HK_{M7} [10]	7983.41	19.23%	15.29%	60.39%

F. Case Study

As a final step, we integrate our linear-regression model (G_{M16}) and the global router (mPFGR) into our placement flow in [3], then perform placement and routing on all 372 benchmarks. Resolving congestion during placement involves first obtaining a congestion estimate (i.e., using one of the previous models), then using that estimate to perform cell/LUT inflation to spread the congestion to less congested regions. The results are shown in Table X. When congestion is not addressed, the Vivado router is unable to route 225 benchmarks. However, only two benchmarks fail to route when using either our congestion-estimation model or the global router. There is no significant difference in final routed wirelength. However, the runtime for the Vivado router is 19% less when using our congestion-estimation model compared to the global router. Fig. 4 shows that our model is consistently faster than the global router. On average, the machine-learning model proposed in this work is 291x faster than the global router as a standalone congestion-estimation technique.

TABLE X: G_{M16} vs. mPFGR vs. No Estimation

Congestion Method	#Failures	Routed-WL (Norm.)	Router Runtime (Norm.)	Placer Runtime (Norm.)
G_{M16}	2	1.00x	1.00x	1.00x
mPFGR [1]	2	1.00x	1.19x	1.17x
No Estimation	225	1.03x	3.15x	0.47x

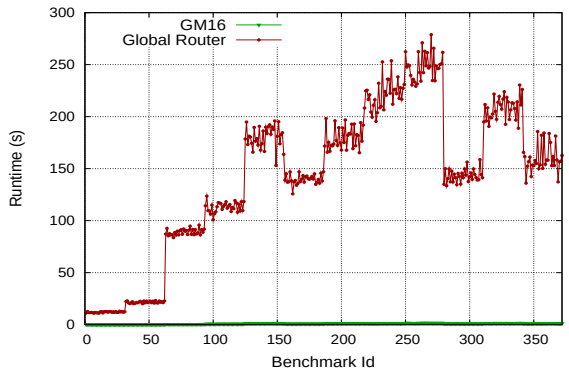


Fig. 4: G_{M16} and mPFGR Runtime (s)

V. Conclusions And Future Work

In this paper, a machine-learning model for estimating congestion during placement has been presented. Based on linear regression, the model employs several relevant features to capture the underlying congestion problem. The performance of the model was tested using 372 benchmarks provided by Xilinx Inc. The experimental results show that the model achieves a 25% improvement in prediction accuracy compared to the machine-learning model in [10]. The performance of the model also performs favorably with that of a global router, as both congestion estimation methods result in the same number of routable placements, and both produce placements with the same post-routing wirelength. However, the runtimes of the two algorithms differ significantly, with our model running, on average, 291x faster. Our future work will focus on applying deep learning to the FPGA congestion-estimation problem.

References

[1] Z. Abuowaimer, D. Maarouf, T. Martin, J. Foxcroft, G. Grewal, S. Areibi, and A. Vannelli. GPLace3.0: Routability Driven Analytic Placer for UltraScale FPGA Architectures. *ACM Transaction on Design Automation of Electronic Systems*, In Press, June 2018.

[2] S. Chen and Y. Change. FPGA Placement and Routing. In *International Conference on Computer Aided Design*, pages 914–921. ACM, 2017.

[3] Z. Dai and D. Banerji. Routability Prediction for Field Programmable Gate Arrays with a Routing Hierarchy. In *Intl' Conference on VLSI Design*, pages 85–90, 2003.

[4] F. P. et al. Scikit-learn: Machine Learning in Python. *J. Mach. Learn. Res.*, 12:2825–2830, Nov. 2011.

[5] G. Grewal, S. Areibi, M. Westrik, Z. Abuowaimer, and B. Zhao. Automatic Flow Selection and Quality-of-Result Estimation for FPGA Placement. In *24th Reconfigurable Architectures Workshop*, pages 115–123, Orlando, Florida, USA, May 2017.

[6] P. Kannan, S. Balachandran, and D. Bhatia. fGREP - Fast Generic Routing Demand Estimation for Placed FPGA Circuits. In *International Conference on Field-Programmable Logic and Applications*, pages 37–47. Springer-Verlag, 2001.

[7] P. Kannan, S. Calachandran, and D. Bhatia. On Metrics for Comparing Interconnect Estimation Methods for FPGAs. *IEEE Transactions on VLSI*, pages 381–385, April 2004.

[8] R. Pattison, Z. Abuowaimer, S. Areibi, G. Grewal, and A. Vannelli. Invited Paper: GPlace - A Congestion-aware Placement tool for UltraScale FPGAs. In *Int' Conference on Computer Aided Design*, pages 1–7, Austin, Texas, November 2016.

[9] C. Pui, G. Chen, W. Chow, K. Lam, P. Tu, H. Zhang, E. Young, and B. Yu. RippleFPGA: A Routability-driven Placement for Large-Scale Heterogeneous FPGAs. In *International Conference on Computer-Aided Design*, pages 1–8, 2016.

[10] 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.

[11] J. Swartz, V. Betz, and J. Rose. A Fast Routability-driven Router for FPGAs. In *Int' Sym. on FPGAs*, pages 140–149. ACM, 1998.

[12] Xilinx. "A Tcl-based CAD Tool Framework for Xilinx's Vivado Design Suite". https://github.com/byuccl/tincr.

[13] Xilinx. ISPD 2016 Routability-Driven FPGA Placement Contest. http://www.ispd.cc/contests/16/ispd2016_contest.html. [accessed 2017-03-17].

[14] Xilinx. "UltraScale Architecture Configurable Logic Block User Guide". http://www.xilinx.com/support/documentation/user_guides/ug574-ultrascale-clb.pdf.

[15] S. Yang, A. Gayasen, C. Mulpuri, S. Reddy, and R. Aggarwal. Routability-Driven FPGA Placement Contest. In *International Symposium on Physical Design*, pages 139–143. ACM, 2016.

[16] 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.

[17] Y. Zhuo, H. Li, and S. Mohanty. A Congestion Driven Placement Algorithm for FPGA Synthesis. In *Field Programmable Logic and Applications*, pages 683–686, September 2006.