electronics

MDPI

Article

Congestion Prediction in FPGA Using Regression Based Learning Methods

Pingakshya Goswami and Dinesh Bhatia *

Department of Electrical and Computer Engineering, The University of Texas at Dallas, Richardson, TX 75080, USA; pingakshya.goswami@utdallas.edu
* Correspondence: dinesh@utdallas.edu

Abstract: Design closure in general VLSI physical design flows and FPGA physical design flows is an important and time-consuming problem. Routing itself can consume as much as 70% of the total design time. Accurate congestion estimation during the early stages of the design flow can help alleviate last-minute routing-related surprises. This paper has described a methodology for a post-placement, machine learning-based routing congestion prediction model for FPGAs. Routing congestion is modeled as a regression problem. We have described the methods for generating training data, feature extractions, training, regression models, validation, and deployment approaches. We have tested our prediction model by using ISPD 2016 FPGA benchmarks. Our prediction method reports a very accurate localized congestion value in each channel around a configurable logic block (CLB). The localized congestion is predicted in both vertical and horizontal directions. We demonstrate the effectiveness of our model on completely unseen designs that are not initially part of the training data set. The generated results show significant improvement in terms of accuracy measured as mean absolute error and prediction time when compared against the latest state-of-the-art works.

Keywords: routing congestion; machine learning; EDA prediction

Citation: Goswami, P.; Bhatia, D. Congestion Prediction in FPGA using Regression Based Learning Methods. *Electronics* **2021**, *10*, 1995. https://doi.org/10.3390/electronics10161995

Academic Editor: Joo-Young Kim

Received: 25 July 2021
Accepted: 13 August 2021
Published: 18 August 2021

1. Introduction

Advances in manufacturing technology and computer-aided design (CAD) tools have resulted in field-programmable gate arrays (FPGAs) that can map several million gate size designs. As the device size increases and the designs become more complex, the effort required to successfully map designs on large devices also increases substantially. Figure 1 shows a modern CAD design flow for FPGAs. The use of High-Level Synthesis (HLS) tools for FPGA-based flows is becoming common as it permits easy conversion of software descriptions into fairly accurate clocked hardware designs. HLS based design methodologies enable designers to focus on algorithmic variations of high-level programs to obtain the best micro-architectural trade-offs. HLS reduces the overall design time as it allows the generation of different versions of a functionally equivalent design without modifying the behavioral code. HLS is typically supported by *design space exploration* that enables a designer to pick one of many functionally equivalent designs that are synthesized from a behavioral description. HLS combined with the *design space explorer* consumes substantial time from the overall design time. Another step that consumes the most amount of time is routing, responsible for connecting the electrically equivalent signals using the routing resources available on an FPGA. It is not uncommon for very high chip utilization designs to fail in producing successful final wiring. Usually, the failed routing requires an iterative design cycle where the placement has to be adjusted to create congestion-free regions to enable wiring. Estimation of interconnect in the local regions of FPGAs is of great interest and has been studied extensively. In Section 2, we discuss various attempts and approaches that span from early-stage empirical models to analytical and statistical models and, currently, machine learning-based prediction models.

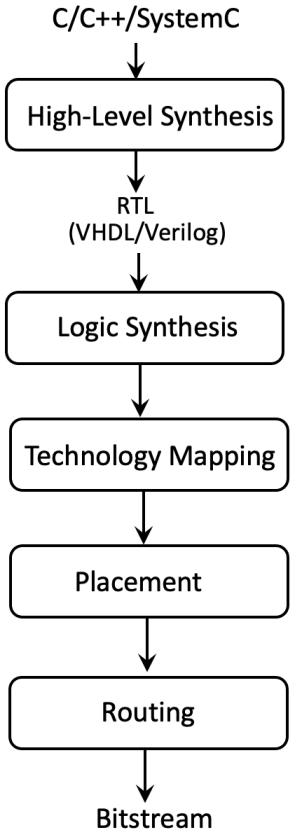

Figure 1. FPGA Design Flow.

Machine Learning (ML) has found its application in various Electronic Design Automation (EDA) problems, and has been used to make early prediction on various quality related parameters. In EDA, ML finds its applications in yield prediction [1], timing [2,3] and power optimization [4], auto-tuning of CAD tool parameters [3], and routing congestion estimation [5–7]. Learning from prior designs and predicting performance and closure issues has immense value in the EDA community. A recent announcement from Xilinx about its ML-suite strongly affirms the idea that ML will play an important role in Electronic Design Automation.

In FPGA physical design flow, routing is the most time-consuming task. It is prevalent for a router to either fail or take a long time to complete the routing, especially for very high logic utilization designs. Locally congested regions can result in failed routing or sub-optimal designs with problems in timing closure. Usually, a designer must replace the design in order to obtain better closure after a failed routing attempt. *Routing Congestion* is measured as the ratio of the number of used routing tracks to the number of available routing tracks. Routing congestion can be measured accurately only after the detailed routing. Traditionally, in ASIC design flows, congestion is estimated after the global routing using statistical models. Since Global Routing-based statistical and probabilistic prediction models do not work well on smaller technology nodes and FPGAs, alternate approaches are needed to predict the routing congestion before the router attempts a complete routing task.

2. Related Literature

Technological advancements and pressures on time to close a design result in many approaches that predict design feasibility during early stages. Routing failures must be predicted before a design is routed in detail. In [8], Liu et al. discuss a Global Route (GR) based routing estimation tool where they take into consideration both local nets inside a cell as well as global nets connecting multiple cells. The authors in [9] introduced a new routing congestion analysis tool called CGRIP that operates on a flexible model of

global routing and models the congestion estimation using Integer Linear Programming. Statistical and probabilistic model-based routing congestion estimation tools are discussed in [10]. Among the other works that employ non-machine learning-based models in order to estimate routing congestion, [11,12] are some of the significant ones. They have converted the routing estimation problem as a graph-based model, and a combination of three factors calculates the routing demand, i.e., number of paths originating from a vertex, routing demand of a bounding box, and routing demand due to large number of terminals in a net [11]. Paper [12] is an enhancement of [11] in which the authors implemented parallel zonal search to reduce prediction time in the high terminal and high fan-out nets. The global routing-based congestion estimation models discussed in [5–9] work perfectly well on larger technology nodes, i.e., greater than 45nm. However, as the process technology size decreases, the miscorrelation between GR-based congestion prediction and actual detailed route congestion increases because of high pin density, complex DRC rules, and increased routing tracks per unit area. This is true for both FPGAs and standard cell-based ASIC design technologies.

To cope with the decrease in technology nodes and ever-increasing complex design rule checking (DRC) rules, researchers are creating Machine Learning models by collecting data from previous design experiences in order to model and predict the behavior of newly produced designs. Early routing congestion prediction is a valuable piece of information that can help minimize unnecessary design iterations. Researchers have framed the routing congestion prediction problem as a supervised learning model. In [5,7], the authors created the routing congestion problem as a binary classification problem. They collected features from placed or routed netlist and predicted whether DRC violations will be present or not because of routing congestion after detailed routing. These papers only predict whether shorts or violations are present after detailed routing. They do not predict the exact value of the congestion on each cell of the design. In [5–7], the researchers addressed the routing congestion estimation problem as a regression model. They do not mention common performance metrics such as R^2, mean square error, etc. The work presented in [6] predicts the presence of DRC violations, but nothing has been mentioned about the location of congested cells or the DRC violated cells. Most of the machine learning-based models can correlate routing congestion and DRC violations much better as compared to the global routing-based routing estimation model.

The works discussed in [5–7] are congestion estimation on standard cell-based ASIC technologies. Routing estimation has also been studied extensively in [13] where a heuristic-based method predicts routing demand that various nets in a design exert on routing resources of an FPGA. Two very recent works propose congestion estimation on FPGA designs [14,15]. In [15], the authors made prediction based on three (feature) parameters. They have reported around 90% accuracy when compared to post-placement routing congestion estimation provided by the Xilinx Vivado Design Suite [16]. However, when compared to the actual congestion reported after detailed routing, the result is around 0.60 R^2 value [14]. This is also an indication that the post-placement congestion estimation provided by Xilinx Vivado is very loose. Moreover, both [14,15] trained and tested their model on ISPD 2016 routing contest benchmark suite [17]. Maarouf et al. [14] also used additional 360 designs for training and testing and reported their accuracy in terms of the R^2 value. It is not clear if it is the best or the average R^2 value. They extracted four features from the post placed and post routed netlists, where the fourth one is a combination of the first three features. Furthermore, for testing purposes, [14] used the standard data splitting into 70–30 ratio, trained the models on the 70% data, and reported the results on the remaining 30%. The testing is not performed on individual designs; instead, the testing data may import many characteristics from the training data elements. Many of these related works assume that exact geometric placement coordinates of various cells are known and assume congestion in estimated post-placement. Balachandran et al. [18] extended the idea of congestion estimation at a higher level of abstraction by presenting methods for estimating the interconnect before the placement. Their approach derives characteristics

from a circuit netlist and then provides wirelength and interconnect estimation. More recently, Zhao et al. [19] have attempted to estimate the routing congestion at the high-level synthesis (HLS) level. They have proposed a machine learning-based method that guides high congestion regions in a design to the high-level source code.

This work addresses a critical need for current designers. Upon completing the placement step, a robust and accurate estimation of congestion in each routing channel would help the designer decide on routing feasibility. Instead of providing a generalized idea about congestion in localized regions, an accurate estimate in every horizontal and vertical channel is a beneficial design aid. Such a fine-grained view of local channel congestion can also help incremental localized placement improvement without routing. In particular, the methodology presented in this work will be highly desirable for the FPGA architectures where the horizontal wiring resources are different from the vertical wiring resources. We leverage the knowledge obtained from previously completed designs to build a machine learning model that helps us in rapidly estimating the localized wiring congestion for random blind designs. We have expressed the routing congestion estimation problem as a regression model. We have outlined the methodology for generating data, for the training of models, for the testing of models, and for integration with commercial CAD tools for comparison.

The rest of the paper is organized as follows. In Section 3, we describe the internal architecture of the Xilinx Ultrascale+ FPGA device. In Section 4, the proposed routing congestion prediction framework is described. In Section 5, the results from the experiments performed are presented. Finally, Section 6 concludes the paper.

3. FPGA Architecture Description

A heterogeneous FPGA device consists of various types of resources inside it, the most prominent of which are CLBs (Configuration Logic Blocks), BRAMs, DSPs, and IO blocks. Each CLB location is surrounded by vertical and horizontal wiring resources spread in the *routing channels*. *Switchbox* and the *Connection Box* allow the wires to connect between horizontal and vertical channels and to connect from the CLB to the wiring resources in the channels, respectively.

We have modeled our routing congestion prediction model for the Xilinx Virtex Ultrascale+ architecture, which is based on TSMC 16nm process node [20]. Figure 2 shows an illustration of the Xilinx Ultrascale+ device. The device is divided into multiple *super logic regions* (SLR). SLRs are only present on devices that use stacked silicon interconnect Technology (SSIT). SSIT, also known as a 2.5D packaging technology, uses *interposers* for interconnection SLRs. In SSIT, multiple dies are packaged together, and each die becomes a super logic region. SLRs contain a 2D array of *fabric sub-regions* (FSRs). FSRs are also popularly called *clock-regions*. In one clock region, the clock is routed so that a single clock supplies each element present in that region. This ensures a near zero-skew clock to each element within the same clock region. In the Xilinx Ultrascale+ devices, each clock region is 60 CLBs tall and is 30 columns wide. The FPGAs are divided into different clock regions so that a particular design is synchronous and meets the timing requirements. For design running in multiple clock domains, each of the IPs required to run at the same clock frequency is placed in the same clock region. The Virtex architecture consists of 8 to 24 clock regions. For logic to communicate between SLRs, the UltraScale architecture employs special tiles in the Clock Region neighboring the abutment of two SLRs. A column of CLBs is removed and replaced with special tiles called Laguna tiles that have dedicated flip flop sites to aid in crossing the SLR divide [21,22]. FSRs contain *configurable logic blocks* (CLBs), and all FSRs are 60 CLBs tall in the UltraScale architecture, but their width will vary depending on the mix of tile types used in its construction. Each clock region or an FSR contains an array of CLBs. Every CLB contains one slice with eight 6-input LUTs and sixteen storage elements. The LUTs in the logic slice is organized as a column with an 8-bit carry chain per CLB, called CARRY8. Wide-function multiplexers combine LUTs to create any function of 7, 8, or 9 inputs or some functions of up to 55 inputs. SLICEL is the name

used to describe CLB slices that support these functions. The LUT in a SLICEM, where the M is for memory, can be configured as a look-up table, 64-bit distributed RAM, or a 32-bit shift register. The CLB for a SLICEL is also referred to as a CLE_L tile, and the CLB for the SLICEM is referred to as a CLE_M tile.

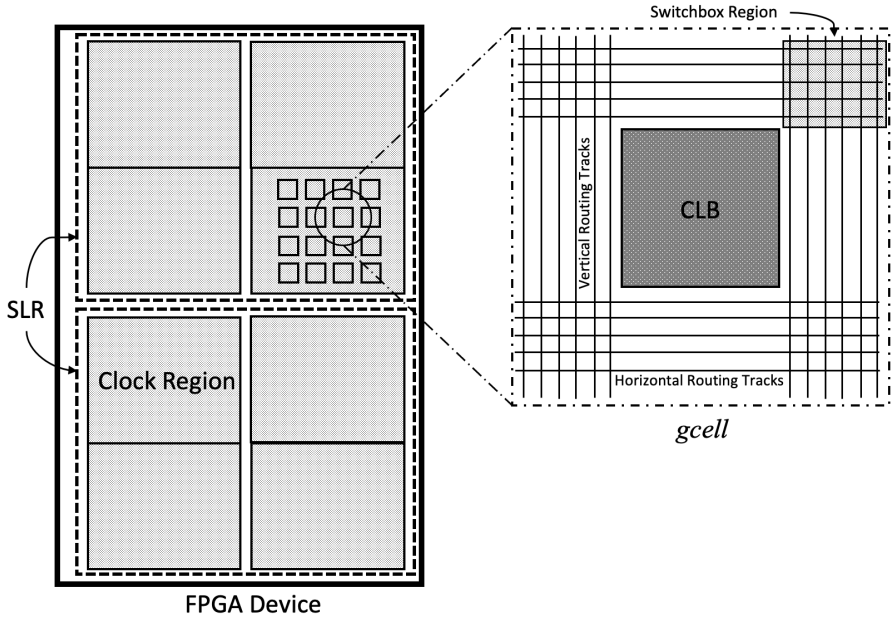

Figure 2. Architecture of Xilinx Ultracale+ FPGAs. The hierarchical architecture has lowest level of granularity at CLB sites. CLB sites are surrounded by horizontal and vertical wiring.

Generally, more than 95% of the FPGA resources and the designs are consumed by LUTs, and Flip Flops present inside the CLB slices. CLBs are surrounded by a limited number of vertical and horizontal routing resources. Commonly, a CLB and its surrounding routing are also called a *gcell*. The Xilinx Vivado routing congestion reporting tool calculates and reports the vertical and horizontal congestion for each CLB block inside an FPGA. In order to correlate and compare the routing congestion predicted by our model with the actual values reported by the Xilinx Vivado routing congestion tool after the detailed routing stage, we also measured the congestion values only for the CLB blocks. Fine-grained prediction of congestion around each CLB is a precious tool for designers as it helps them improve the localized placement to locally adjust or remove over-congested regions. Xilinx Vivado reports horizontal and vertical congestion separately for each *gcell*. It calculates the congestion by taking the ratio of the number of routing tracks available to the number of tracks used. Mathematically, congestion is given by the following.

$$congestion = \frac{number\ of\ tracks\ required}{number\ of\ tracks\ available} \tag{1}$$

A design is generally considered highly congested if it contains a very large number of *gcells* for which its congestion value is greater than 90%.

4. FPGA Routing Congestion Prediction Framework

Accurate prediction of post-route congestion around each CLB is valuable information for designers to ensure a rapid and feasible design closure. Therefore, we have formulated the routing congestion estimation problem as a machine learning-based regression model in order to accomplish precise congestion prediction. Figure 3 illustrates our overall methodology where we have described a flow to *train* a regression model using the features described in the Section 4.2, and then the trained model is used to predict the routing congestion on completely unseen data during the *test* phase of our design flow.

4.1. Prediction Framework Flow

The proposed training and deployment model for our regression-based routing congestion estimation is illustrated in Figure 3. Two design flows are used for (i) creating the model for predicting the routing congestion and (ii) testing and predicting congestion using the trained model. It is significant to note that the testing flow uses never-before-seen designs and is, thus, utterly blind to any data properties that belong to the test design set.

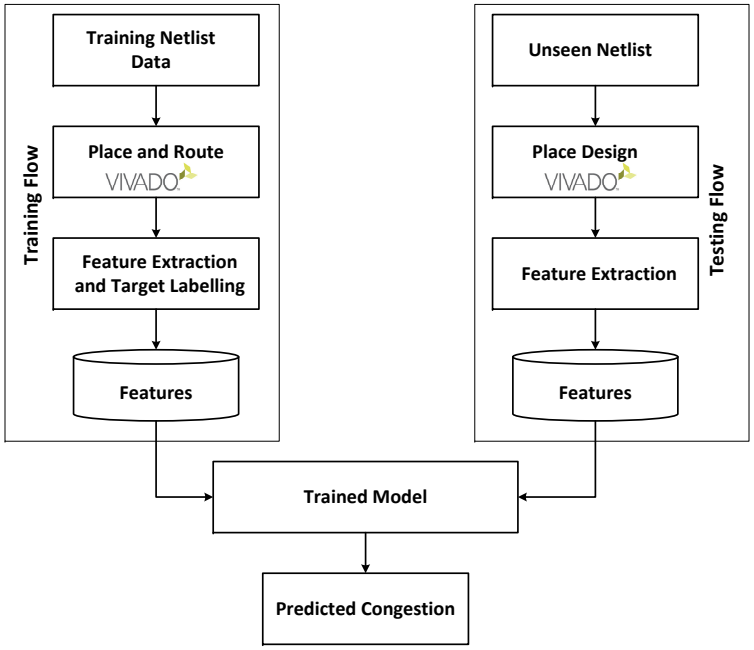

Figure 3. Design flow for congestion prediction model training and model-based congestion prediction.

The *training flow* in Figure 3 takes a design netlist as an input. The design is placed and routed for a selected FPGA device using the Xilinx Vivado 2018.3 toolset. As we have stated later, in our experiments, we have used Xilinx Ultrascale Virtex (Xilinx XCVU095) devices for mapping the designs. This placement and routing step results in determining actual post-route horizontal and vertical congestion for each CLB site (*gcell*). These actual values of horizontal and vertical congestion around each used CLB are retained as a *label* for creating the model. Using the feature extraction methodology described in Section 4.2, we extract nine features for each of the utilized CLB sites for the mapped design mapped. The extracted features and corresponding labels are used for building regression-based models for the prediction of congestion. The regression models are described in Section 4.3, and we have used CometML [23] machine learning platform for our experiments.

The *testing flow* in Figure 3 takes the netlist of the design under test and performs placement by using the Xilinx Vivado toolset. Again, we extract the same *nine* features for each of the utilized CLB sites for the design mapped on the Xilinx XCVU095 device. The feature map is then passed through the trained model to predict the routing congestion for each utilized CLB site or the *gcell*. The offline training of our model is performed using the data extracted from the initial examples of the ISPD 2016 FPGA Placement benchmarks [17]. As illustrated in Figure 4, E1–E4 are used for extracting the training and cross-validation data. Finally, the correctness and robustness of our trained model are verified by using (unseen/blind) test data during the testing phase of routing estimation (prediction). We have used twelve of the ISPD 2016 FPGA Placement contest benchmarks to final test our proposed methodology. It is noteworthy that the testing phase uses completely unseen data that were never exposed during the training phase of the model. We have verified the correctness of our prediction by computing the mean absolute percentage error against the actual congestion reported by the industry standard Xilinx Vivado router.

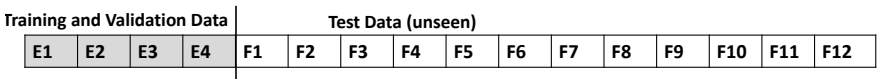

Figure 4. Experimental data. The training and validation data, E1–E4, are used to create a model. Prediction model is tested on completely unseen data, F1 through F12. ISPD 2016 placement contest benchmarks are used for experiments.

4.2. Feature Extraction for Placed Netlists

The congestion prediction model takes input in the form of features that are extracted from the placed netlists. The features must capture the design and device specific characteristics, and we are interested in the smallest set of most essential features that will help us in quick training of the model and the prediction of the congestion. As in the case of any machine learning application, the selection of features also known as *Feature Engineering* is one of the most crucial and challenging stages. Correct selection of features allows us to build a robust model for accurate congestion prediction, and too many redundant features may result in noise and *overfitting*.

We explored many features related to FPGA device architecture and the mapped design characteristics. These included (i) *super logic region* ID or the distribution of placed logic on a specific *super logic region* (SLR), (ii) placement within specific *clock-region*, (iii) *gcell* belonging to a specific CLE type, and (iv) total number of LUTs, Flip-Flops, and Multiplexors in a CLB. The CLE type is a *categorical* feature and does not assume any numerical value. However, the functional usage of each CLE type is very different, and we expected that the routing demand would differ around each different CLE type. In addition to these device specific features, we also explored features that we believed will have impact on routing congestion as a result of mapping of a design on a specific device. The following is the list of important features for each of the utilized CLBs/*gcell*s described in Section 3. We have numbered the features as f_1 to f_9. Along with each feature's description, we also describe our intuition for selecting the feature.

i. Number of Utilized Cells (f_1): A Xilinx Ultrascale+ CLB slice consists of four different types of cells, viz., LUT, Flip-Flops, Multiplexers, and Carry Chains. One CLB has multiple units of each cell type. The number of cells utilized by a net inside each CLB tile plays an important role in measuring the congestion around each CLB block. When more and more cells within a CLB tile are used, more wiring resources would clearly be required to interconnect them. Based on this intuition, we calculated the number of cells used inside each of the CLB slices and used it as a vital parameter in our ML based prediction model. If more cells are used, the probability of congestion around the CLB increases proportionately.

ii. Number of Pins (f_2): Each cell in the CLB slice contains a certain number of pins. For example, a H6LUT contains eight pins (6 inputs and 2 outputs), a mux contains four pins, and a flip-flop contains five pins. When pins are connected to wires, they are an indicator of local congestion inside the CLB tile. We keep track of the number of used pins in each CLB tile as a useful feature.

iii. Number of incoming/outgoing nets (f_3 and f_4): If a net has pins spread across multiple CLB slices all over the FPGA, it can be considered as a multi-CLB net. Each multi-CLB net can be covered by a bounding box. The number of multi-CLB incoming and outgoing nets passing through a CLB slice provides us with the number of bounding boxes that crossed that particular CLB. The greater the number of bounding boxes passing through a CLB slice, the higher the congestion results. Hence, the number of bounding boxes passing through a CLB slice is an important feature for estimating the routing congestion around it.

iv. Number of Local/Buried nets (f_5): Local nets or buried nets represent nets where all the cells are present in the same CLB slice. Even if all the cells are placed inside one CLB, the wires use the routing tracks present in the *gcell* to route the signals. Hence, this feature also contributes to routing congestion in the *gcell*.

v. Location of the *gcell* inside the FPGA (f_6 and f_7): The absolute location of a *gcell* is not very relevant. When a group of cells is placed in a localized region, every cell in the group impacts congestion of other cells in the group. Analysis shows that the location of a tile inside the FPGA plays one of the crucial roles in routing congestion estimation, as a tile in an already congested region will experience greater congestion. This is because the neighboring *gcell*s of a *gcell* play an important role in the congestion. Gcells which are surrounded by high cell density or high pin density gcells will have high routing congestion.

vi. LUT (f_8) and Flip Flop (f_9) Utilization: Each CLB or *gcell* consists of 8 LUTs and 16 FFs. This feature represents the percentage of LUTs and FFs used inside a *gcell*. These features also contribute to congestion, but their effect is very low than compared to the other features discussed above.

We selected many features but, after rigorous experiments and passing the features through a feature importance measuring tool [24], we discarded extra features. We used the ELI5 [24] Python package to analyze the feature importance on the regressor models. ELI5 computes feature importance by measuring how the accuracy of a model decreases if one of the features is not available. This method is called "*Permutation Importance*". Since removing processes and retraining processes are computationally intensive, ELI5 replaces the feature column with random noises in the test dataset. ELI5 inserts noise by repeatedly shuffling the value of that feature with some other row's value. All of the nine features were found to be important, and their importance in the decreasing order is listed as follows: $\{f_5, f_1, f_6, f_f, f_2, f_8, f_9, f_3, f_4\}$. Table 1 illustrates an example of a data-frame that depicts values associated with various features. All of the useful features are listed in the columns of Table 1, and rows list four sample data elements for four *gcells*. The discarded features include the following:

1. *Super Logic Region* `id`;
2. Placement of logic within a specific *clock-region* in an SLR;
3. CLE type;
4. Total number of LUTs, Flip-Flops, and Multiplexors in a CLB.

The discarded features are related to the characteristics of the FPGA device used. The regional device specific characteristics that include the CLE type, *clock-region*, and SLR region do not have any impact on the model quality or prediction quality.

Table 1. Sample dataframe showing the features and four example data points.

f_1	f_2	f_3	f_4	f_5	f_6	f_7	f_8	f_9
2	8	6	2	7	3	86	12.5	6.25
8	41	33	8	24	4	86	62.5	18.75
5	22	17	5	19	4	88	25	18.75
4	21	17	4	18	4	90	25	12.5

4.3. Creation and Training of Regression Model

Features f_1 to f_9 provide important data related to the characteristics of mapping of a design on an FPGA design. In order to create a model for predicting the local channel congestion, we need the data and corresponding labels. We have placed and routed the benchmark circuits by using Xilinx Vivado tools. As stated in Section 4, the placement and routing step results in determining the actual post-route horizontal and vertical congestion for each CLB site (*gcell*). These actual values are used as training labels for the prediction models. We have trained our model by using four different regression models. The training data generated in Section 4.4 is divided into 70% for training and 30% for validation. We also used four-fold cross validation by using all of the databins generated from four example designs used for training. Training is conducted locally on an Intel Core I7

3.6 GHz octacore processor. We also trained and validated the model on Amazon AWS Sagemaker [25] for cloud based training and deployment by using T2 medium instance. We used Comet ML [23], an open source ML tool, for training and hyperparameter tuning. A brief theoretical description of the regression models used is described here.

i. **Linear Regression**: Linear regression is a method to find relationship between two continuous variables, where one is independent and the other is dependent. Linear regression tries to create a statistical relationship which may not be always deterministic. Linear regression tries to obtain a straight line which shows the relationship between the independent and the dependent variable. For most of the linear regression models, the error is measured by using least squared error. A linear regression line has an equation of the form $Y = a + bX$, where X is the independent variable and Y is the dependent variable. The slope of the line is b, and a is the intercept, which are both determined during training of the model.

ii. **Random Forest Regression**: Random forest is a supervised learning method that uses ensemble methods for learning. Ensemble learning is a technique that combines multiple weak learners to create a strong learner. In random forest, these weak learners are decision trees. The outputs from the individual learners are averaged in order to generate the final output. Random forest belongs to the bagging class of learners where, during training, data samples are selected at random without replacement. Bagging makes each model run independently and then aggregates the outputs at the end without preference to any model. Due to the introduction of the bagging method, the chances of overfit are lower in random forest. Random forest is a very fast method of training because each tree can be trained in parallel, and the inputs and outputs of each individual tree are not related to another. To summarize, the Random Forest Algorithm merges the output of multiple decision trees to generate the final output.

iii. **Multivariate Adaptive Regression Splines (MARS)**: Multivariate adaptive regression splines (MARS) [26] is an algorithm that automatically creates a piecewise linear model which creates a non-linear model by combining multiple small linear functions known as steps. In MARS, non-linearity is introduced by using step functions. There are no polynomial terms in the MARS equation. Equation (2) shows the linear step wise equation:

$$y_i = \beta_0 + \beta_1 C_1(x_i) + \beta_2 C_2(x_i) + \cdots + \beta_d C_d(x_i) + \epsilon_i \quad (2)$$

MARS is an adaptive procedure for regression and is well suited for high-dimensional problems (i.e., a large number of inputs). The bends in the step functions are known as "*knots*". If there are multiple knots present in the MARS equation, it may result in overfitting.

iv. **Multi Layer Perceptron based Regression (MLP**: MLP based regression models are made up of multiple perceptrons also known as neurons. This type of model falls into the feedforward class of artificial neural networks. Generally, ANNs are used for classification. However, they can also be used for regression. MLP based models are trained using backpropagation algorithm. Each network consists of an input layer of neurons, few hidden layers, and an output layer. The output of each neuron is linear. In order to make the output non-linear and to simulate the real world behavior, activation functions such as "*tanh*" and "*sigmoid*" are added.

The regression models have a varying degree of complexity and take different times to train a prediction model. Out of the four regression models presented in the Section 4.3, linear regression was easily discarded because of its inferior accuracy. This is because of the high dimensional feature space; the linear regression model is not able to converge. Figure 5 shows the training time for the four models. The *y-axis* is a log-scale time expressed in minutes. Linear Regression has the fastest training time while Neural Network based MLP regressor is the slowest. Random forest divides the training datasets randomly, and each sub-dataset is independently processed through a prediction classifier such as a decision

tree; hence, it is relatively fast compared to other classifiers. Due to the presence of multiple inputs, hidden and output layers, and the slow convergence time of backpropagation, MLP is substantially slower than others.

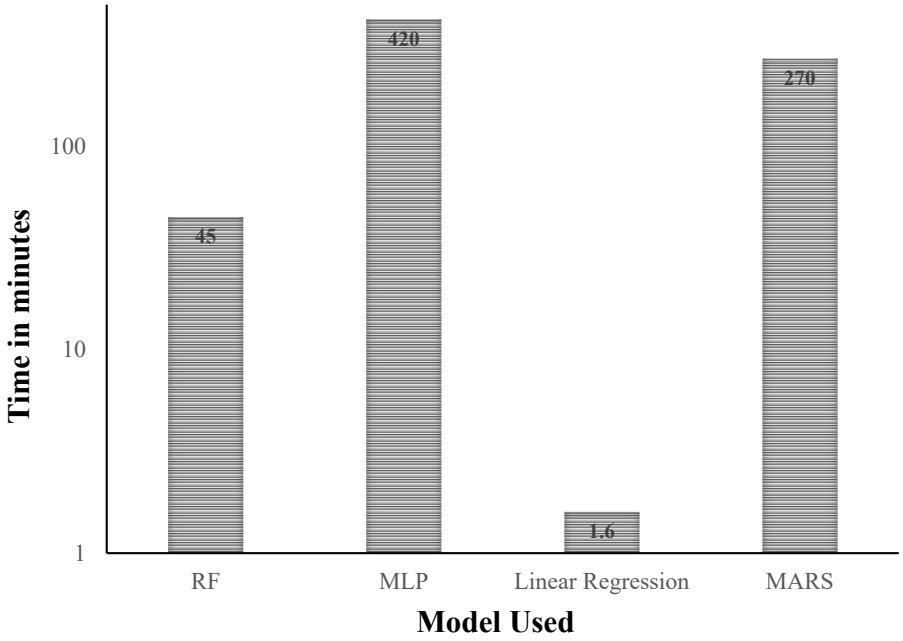

Figure 5. Model training time for different regression methods.

4.4. Generation of Training Data

In order to extract information and various features, we have divided the FPGA into a number of Global Routing cells or *gcells* which act as databins for the features. Figure 2 shows the architecture of one databin. Although we are using only four benchmarks for training our model, the benchmarks are large and diverse. Combining all the utilized gcells present in the four training benchmarks can generate around 160,000 databins. Figure 6 shows the post-routed floorplan of the largest of the four examples (FPGA Example 4) inside the Xilinx XCVU095 device. It can be observed from Figure 6 that Example 4 is highly dense and congested and covers more than 80% of resources present inside the FPGA device. Table 2 shows the statistics of the resources consumed by the four training datasets, while we have shown the total number of units of each resource present inside the XCVU095 device in Table 3. The number of utilized gcells is also noted in the databins column.

Table 2. Statistics of the training data. The examples E1–E4 are from the ISPD 2016 placement contest benchmarks.

	Number of Units Used			
Design Name	**FF**	**LUT**	**BRAM**	**Databins**
EXAMPLE 1	1260	1968	2	325
EXAMPLE 2	237,012	289,036	384	44,565
EXAMPLE 3	175,585	244,435	495	46,161
EXAMPLE 4	344,295	447,021	1114	66,728
Total				157,779

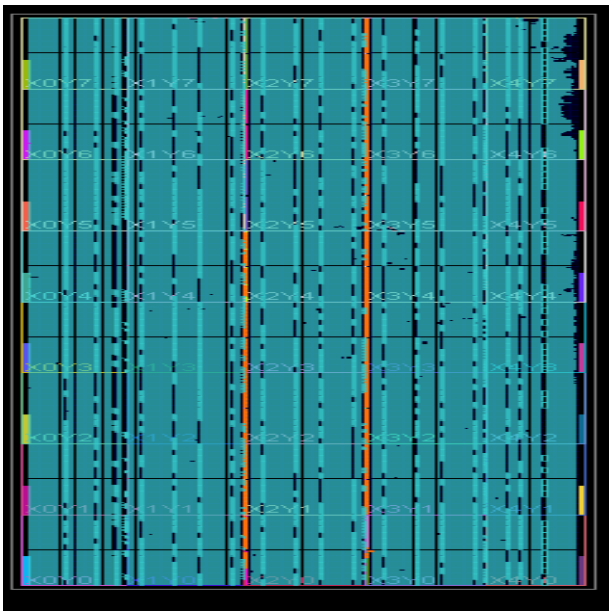

Figure 6. Routed Floorplan of FPGA Example 4 design, which consists of around 67k databins.

Table 3. Physical resources in an FPGA device—Xilinx XCVU095 FPGA.

LUT	FF	BRAM
537,600	1,075,200	1728

5. Experiments and Results

We have trained and tested our model on an Intel Core I7 3.6 GHz octa-core processor with 16GB RAM. The description of the training dataset is described in Section 5. We can extract more than 160k databins that are of heterogeneous and varied nature. We tested our model on the 2016 FPGA placement contest benchmarks. The prediction and validation are performed on both PC and AWS Cloud using free version of Sagemaker, which allows us to use low-speed T2 medium instance for inference. The characteristics of the test benchmarks used are mentioned in Table 4. Column one lists the benchmark names, column two lists the number of nets in each benchmark, column three lists the total number of cells (LUT and FF elements) in each benchmark, and column four lists the number of databins or *gcells* in each benchmark. The designs shown in Table 4 are completely unseen and were not part of the training/validation set. The design size varies from about 10K to 66K databins.

For prediction, we partitioned the placed netlists into databins where each databin consists of one CLB tile also known as the *gcell*. We extracted the same high impact and most relevant f_1 tof_9 features which are same as the training dataset. We also routed the placed netlists to obtain the actual value of routing congestion for each gcell, which are used for evaluation and comparison. We trained our model using the data listed in Table 2 by using an open-source tool called Comet ML [23] and compiled our code on Anaconda Python interpreter by using Python 3.5.6 version. The hyperparameters and the weights generated by Comet ML is used in the remaining 12 test benchmarks to predict congestion values. Around 70% of the training time shown in Figure 5 is used in hyperparameter tuning of the models, while the actual training takes 30–40% of the total training time.

Table 4. Statistics of the testing data. The examples FPGA1 to FPGA12 are from the ISPD 2016 placement contest benchmarks.

Benchmark	# of Nets	# of Cells	#of Databins
FPGA1	104,061	103,958	10,161
FPGA2	165,380	163,780	17,326
FPGA3	424,679	416,877	40,332
FPGA4	426,857	419,056	42,281
FPGA5	431,127	423,332	50,115
FPGA6	658,627	649,435	62,369
FPGA7	684,910	675,699	66,194
FPGA8	722,314	714,515	66,814
FPGA9	876,754	867,954	67,200
FPGA10	786,967	778,207	64,821
FPGA11	822,378	816,034	66,522
FPGA12	966,389	959,025	66,063

5.1. Analysis of Accuracy

In this section, we compared the performance of the four regression models. The effectiveness of a regression model is measured using the R-squared value, which is known as the coefficient of determination. R-squared is a statistical measure for measuring how close the data are to the fitted regression line.

R^2 is defined in the Equation (3). N is the number of samples used in the regression, y_i is the actual value of sample i, $\hat{y}_i$ is the predicted value using a regression model based on observations, and $\bar{y}_i$ is the mean of the actual values.

$$R^2 = 1 - \frac{\sum_{i=1}^{N}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{N}(y_i - \bar{y}_i)^2} \tag{3}$$

We measured the performance of our models by using R^2 values. The R^2 is always ranged between 0 and 1, 0 being the worst fit and 1 being the best. Practically, a R^2 value of 1 is impossible if we try to predict on completely unseen data. However, a R^2 value of 1 is possible during training, but it may indicate a highly overfitted model. Any R^2 value above 0.8 for the type of data we have is considered a good value. We calculated the R^2 value for congestion on both vertical and horizontal directions by using each of the four models, and the results are shown in Figure 7.

As it can be observed from Figure 7, random forest and MLP regression, which use gradient descent based neural networks, produce the best R^2 value. This is because random forest is an ensemble method in which each tree is trained by using a different method, and the random forest combines the output of each tree to give us a prediction. Due to the presence of a large number of neurons in the hidden layers and robust tuning of hyperparameters, the accuracy of MLP regressor is almost equal to that of random forest. We achieved an average R^2 value of above 85% for both vertical and horizontal routing congestion by using random forest and above 82% by using MLP based regressor. MARS is a piecewise linear regression which works well on training data, but it fails in unseen data because of overfitting. From our experiments, we found out that Linear Regression does not work for routing congestion because of the large variance and the variety of features and data we have. Figure 7 shows the graphical representation of average R^2 value for each of the 12 test designs. We selected random forest and multi-layer perceptron for building prediction models to estimate localized congestion in FPGA routing channels.

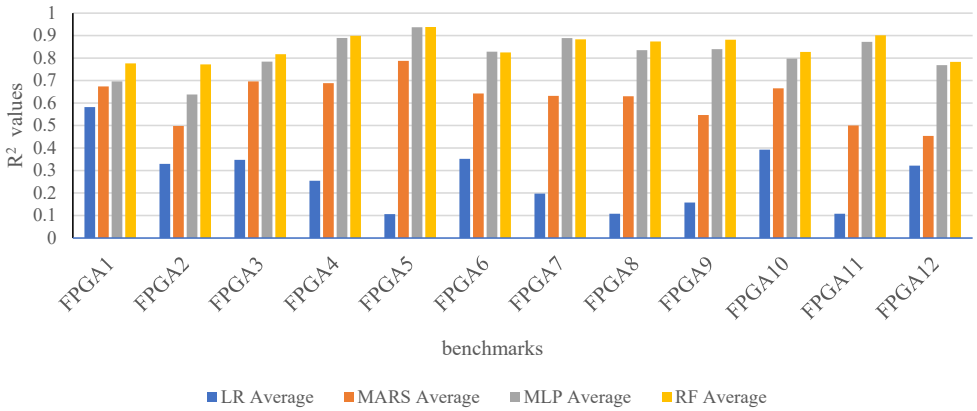

Figure 7. Average R^2 value for the 12 benchmarks using 4 regression models. LR: Linear Regression; MARS: Multivariate Adaptive Regression Splines; MLP: Multi Layer Perceptron; RF: Random Forest.

We have used *Mean Absolute Percentage Error* (MAPE) as a precision indicator to characterize the error in prediction. This approach is consistent with other research reported in [14]. The MAPE is stated in Equation (4), where N is the number of samples used in the regression, y_i is the actual value of sample i, and $\hat{y}_i$ is the predicted value using a regression model based on observations.

$$MAPE_{est} = \frac{1}{N}\sum_{i=1}^{N}\left|\frac{y_i - \hat{y}_i}{y_i}\right| \times 100 \tag{4}$$

We have placed and routed each of the twelve benchmarks by using Xilinx Vivado tools. The post-routing congestion is reported for each channel around CLB site(s) by the Xilinx tools. This forms the set of y_i values in the above equation. We also extract features from the placed designs and pass them through the trained models. The model generates predicted values of congestion in the localized horizontal and vertical channels. These form the $\hat{y}_i$ set of values in the above equation. Each $\hat{y}_i, 1 \leq i \leq N$ is an estimate of the local congestion value, a valuable aid to the designer for being able to observe the congestion value without ever routing the design. As our results illustrate below, $\hat{y}_i$ is a very correct and accurate estimate of the actual congestion value. MAPE is computed for each design when N databins are used for generating features. In Table 5, we have reported the MAPE values of both horizontal and vertical congestion by using MLP and random forest based regression method. As observed from the Table 5, we achieved average MAE of 4.1% and 4.4% for all the designs using Random Forest Regressor and MLP Regressor, respectively.

In order to illustrate the accuracy of our prediction models, we have plotted congestion density maps, also called *heat maps*, for a representative set of designs. Figure 8 shows the comparison of the congestion heat map (average of horizontal and vertical congestion). The illustration was created using MATLAB and is based on the real reported congestion value as well as the predicted congestion value. The Xilinx congestion reporting tool presents horizontal and vertical congestion values as a percentage value. Since our training process makes use of the labels generated by the Xilinx tool, the scale in figure map represents congestion values from 0 to 100%. There is marked resemblance between the plots in the left column with the plots in the right column. In order to further illustrate the accuracy at local channels where congestion percentages are evaluated, we have shown a graphical comparison between the actual and the predicted congestion values in Figure 9. Since each benchmark mapping on an FPGA contains several thousand *gcells*, we have illustrated the congestion values at a subset of *gcells*. The graphical representation is illustrated for the same three benchmarks that are also illustrated in congestion density plots in Figure 8.

Table 5. MAPE values for predicted congestion values for Random Forest and Multi-Layer Perceptron models.

	Random Forest		MLP	
Design	**Vertical**	**Horizontal**	**Vertical**	**Horizontal**
FPGA1	1.689	4.649	4.894	1.910
FPGA2	2.186	1.721	2.159	2.805
FPGA3	3.370	2.998	3.237	3.653
FPGA4	4.638	3.949	4.117	4.746
FPGA5	5.311	4.870	4.965	5.250
FPGA6	3.591	3.634	3.694	3.379
FPGA7	4.745	4.847	4.650	4.760
FPGA8	4.498	3.886	4.349	5.395
FPGA9	5.263	6.103	4.863	5.467
FPGA10	3.868	3.859	4.117	4.169
FPGA11	5.373	5.318	6.157	5.859
FPGA12	7.994	5.318	5.365	8.205

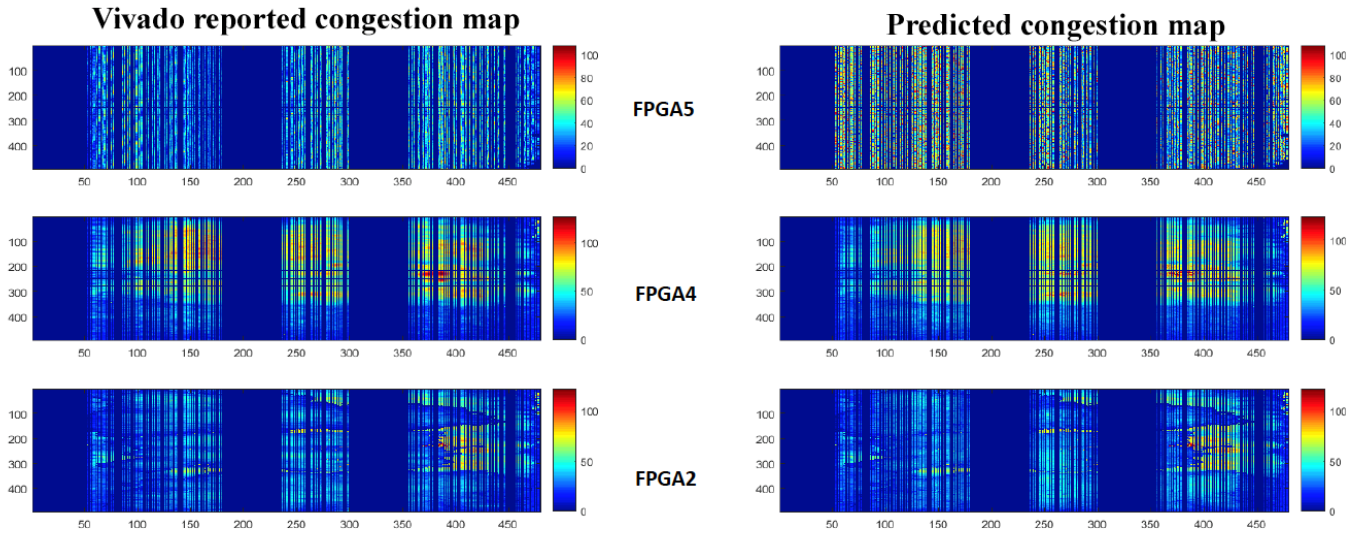

Figure 8. Visual comparison of congestion density plots between actual congestion reported by Xilinx tools and the congestion predicted by the prediction model.

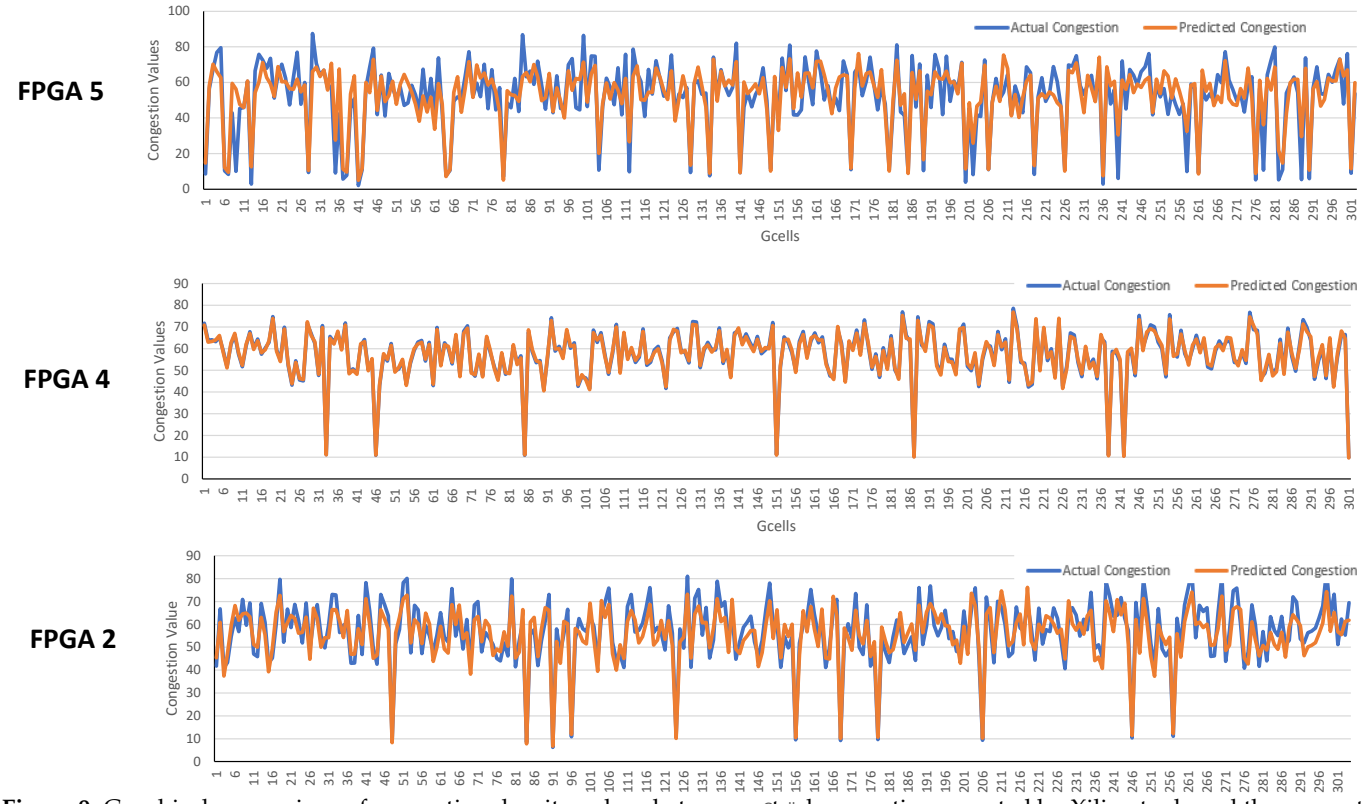

Figure 9. Graphical comparison of congestion density values between actual congestion reported by Xilinx tools and the congestion predicted by the prediction model.

5.2. Discussion

Early estimation of routing congestion is important to ensure rapid convergence and routability of a given design. Our congestion prediction models make use of nine very important features in order to predict the resulting congestion for a given design. The predicted congestion value is highly localized and helps providing an early assessment of routability of a given design. Many previous studies related to congestion and routability estimation have been stated in Section 2. Two closely related works highlighted in [14,15] are worthy of comparison. Reference [14] was later extended in a publication [27]. Pui et al. [15] presented an ML based congestion estimation method that makes use of three features. It was later shown in [27] that only two of the three features used in [15] were actually relevant. The average MAPE reported in [15] is around 8%. The authors of [14] also makes use of a total of three (f_1, f_2, and f_3) features (which can be seen as four features as feature f_3 is further divided into two types) and an extensive training and testing framework in order to produce far superior MAPE values for congestion prediction. Both [14,15] make use of standard training and testing framework where entire data are divided according to the standard and generally accepted rule of 70% for training and 30% for testing. In particular, Maarouf et al. [14] make use of a large set of 372 benchmarks. The 372 designs consist of 12 ISPD 2016 Placement Contest designs and another 360 benchmarks that were generated using a netlist generation tool. The work presented in this paper demonstrates a carefully crafted regression model based congestion prediction method. As described in the previous sections, our model construction was performed by carefully examining the design mapping on an FPGA. The localized features are easily derived from the mapped design and do not require much computation overhead. There is a slight difference for congestion reporting in [14] and our approach. We report congestion in the horizontal and vertical channels around the CLB site. This is consistent with the way congestion is reported by the tools within Xilinx Vivado. Therefore, our prediction result can integrate directly with the Xilinx Vivado tool flow. The congestion value, as reported

in [14], is computed at the switchbox location. Thus, while the prediction MAPE can be compared, their actual interpretation would be slightly different. Our report is consistent with the values reported within the Xilinx tool flow. The average reported MAPE for [14] is 4.69% while our prediction method achieves a MAPE of 4.1%. Likewise, the maximum R^2 value reported in [14] and for our method is 86% and 94%, respectively. Usually, the actual value of congestion would be known after routing. In Table 6, we have show the effectiveness of our model based congestion prediction. Columns 2 and 3 in Table 6 show the congestion prediction executing time in seconds by using the multi-layer perceptron and random forest models, respectively. In order to illustrate the advantage of early prediction, we have also listed the actual routing execution time using Xilinx Vivado 2018.3 in column four of Table 6. Although there is substantial execution time difference between the congestion prediction times reported by the MLP and RF models, the prediction still is orders of magnitude faster than full FPGA routing.

Table 6. Prediction time using ML model vs. Actual Xilinx Vivado 2018.3 Routing Time.

Design	MLP	RF	Actual Routing
	Seconds		Minutes
FPGA1	15.023	0.955	15
FPGA2	73.586	1.542	24
FPGA3	143.336	3.761	24
FPGA4	226.608	4.005	36
FPGA5	276.508	5.038	66
FPGA6	266.521	6.165	84
FPGA7	427.346	6.583	114
FPGA8	368.471	5.985	132
FPGA9	411.258	5.896	141
FPGA10	189.253	5.854	150
FPGA11	288.107	6.309	168
FPGA12	221.788	6.442	192

It can be observed from the results that random forest is the best suitable model for the type of features we have both in terms of accuracy and prediction time. We also observe that we cannot compare the model training times for our method and the ones proposed in [14,15] as the characteristics and size of training data are very different. We have demonstrated that with a mere 160K databins that are derived from the example set $E_1, ..., E_4$ of ISPD Placement Contest benchmarks, we can train a robust model that produces reliable congestion predictions for a large set of unseen benchmark designs. For prediction time, a fair comparison cannot be made since we predict on one design at a time, but [14] predicts on the 30% of the total dataset, which contains a significantly greater number of *gcells* than one single design. For our model, the maximum congestion prediction time using the *random forest* regression model is about 7 s, as observed in the Table 5.

6. Conclusions

In this paper, we have proposed a robust regression based routing congestion estimation method that predicts the congestion after placement in FPGAs. It is on an average around 25 to 50 times faster than Xilinx Vivado based routing calculation tool which reports actual congestion after detailed routing. The following are the major characteristics and contributions of our work:

1. The methodology is to generate a robust congestion prediction model that makes use of easily obtained features from a placed design. The model exhibits an average MAPE of 4.1% and maximum R^2 value of 94% on completely unseen designs. These are the best known and tightest results.
2. The post-placement congestion prediction values provide accurate congestion in horizontal and vertical channels. The report is consistent with the post-route congestion estimation that Xilinx Vivado tools provide after routing. This enables easy integration of our model based prediction in the Xilinx Vivado tool flow.
3. Our methodology demonstrates that a carefully crafted set of features based model can be trained with a relatively small set of training data. Our models were trained with about 160K databins derived out of four benchmark examples. The test data were completely unseen data.

Our models produce highly accurate predictions of congestion. Although the error is very low, we feel that the model accuracy can be improved by training the models by using a larger and more diverse data set. We demonstrated that with four large benchmarks, we can achieve high accuracy for completely unseen test data sets. The accuracy will improve if the training data sets were more diverse. Hyper-parameter tuning has impact on the model accuracy, and we feel that the model can be made more accurate by careful tuning of the hyper-parameters. These are several extensions that can improve the overall ML based design flow. We are working on an automatic CAD parameter suggesting tool using ML, which can be integrated with the congestion estimation tool such that it will automatically tune the tool parameters in order to minimize very high congestion areas after design placement.

Author Contributions: Conceptualization, methodology, formal analysis—P.G. and D.B.; software, validation—P.G.; resources, supervision D.B.; writing of original draft—P.G.; writing review and editing—D.B. Both authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Data Availability Statement: All of the data and related `python` code files for building machine learning models are available for download [28].

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Xu, X.; Matsunawa, T.; Nojima, S.; Kodama, C.; Kotani, T.; Pan, D.Z. A Machine Learning Based Framework for Sub-Resolution Assist Feature Generation. In Proceedings of the International Symposium on Physical Design (ISPD 2016), Santa Rosa, CA, USA, 3–6 April 2016; ACM: New York, NY, USA, 2016; pp. 161–168. [CrossRef]
2. Yanghua, Q.; Ng, H.; Kapre, N. Boosting convergence of timing closure using feature selection in a Learning-driven approach. In Proceedings of the 2016 26th International Conference on Field Programmable Logic and Applications (FPL), Lausanne, Switzerland, 29 August–2 September 2016; pp. 1–9. [CrossRef]
3. Kapre, N.; Ng, H.; Teo, K.; Naude, J. InTime: A Machine Learning Approach for Efficient Selection of FPGA CAD Tool Parameters. In Proceedings of the 2015 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays, Monterey, CA, USA, 22–24 February 2015; ACM: New York, NY, USA, 2015; pp. 23–26. [CrossRef]
4. Chang, W.; Lin, C.; Mu, S.; Chen, L.; Tsai, C.; Chiu, Y.; Chao, M.C. Generating Routing-Driven Power Distribution Networks with Machine-Learning Technique. *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst.* **2017**, *36*, 1237–1250. [CrossRef]
5. Chan, W.T.J.; Ho, P.H.; Kahng, A.B.; Saxena, P. Routability Optimization for Industrial Designs at Sub-14Nm Process Nodes Using Machine Learning. In Proceedings of the 2017 ACM on International Symposium on Physical Design, Portland, OR, USA, 19–22 March 2017; ACM: New York, NY, USA, 2017; pp. 15–21. [CrossRef]
6. Zhou, Q.; Wang, X.; Qi, Z.; Chen, Z.; Zhou, Q.; Cai, Y. An accurate detailed routing routability prediction model in placement. In Proceedings of the 2015 6th Asia Symposium on Quality Electronic Design (ASQED), Kuala Lumpur, Malaysia, 4–5 August 2015; pp. 119–122. [CrossRef]
7. Tabrizi, A.F.; Darav, N.K.; Xu, S.; Rakai, L.; Bustany, I.; Kennings, A.; Behjat, L. A Machine Learning Framework to Identify Detailed Routing Short Violations from a Placed Netlist. In Proceedings of the 55th Annual Design Automation Conference, San Francisco, CA, USA, 24–29 June 2018; ACM: New York, NY, USA, 2018; pp. 48:1–48:6. [CrossRef]

8. Liu, W.; Wei, Y.; Sze, C.; Alpert, C.J.; Li, Z.; Li, Y.-L.; Viswanathan, N. Routing congestion estimation with real design constraints. In Proceedings of the 2013 50th ACM/EDAC/IEEE Design Automation Conference (DAC), Austin, TX, USA, 29 May–7 June 2013; pp. 1–8.
9. Shojaei, H.; Davoodi, A.; Linderoth, J.T. Congestion analysis for global routing via integer programming. In Proceedings of the 2011 IEEE/ACM International Conference on Computer-Aided Design (ICCAD), San Jose, CA, USA, 7–10 November 2011; pp. 256–262. [CrossRef]
10. Saeedi, M.; Zamani, M.S.; Jahanian, A. Evaluation, Prediction and Reduction of Routing Congestion. *Microelectron. J.* **2007**, *38*, 942–958. [CrossRef]
11. Kannan, P.; Balachandran, S.; Bhatia, D. fGREP—Fast Generic Routing Demand Estimation for Placed FPGA Circuits. In *Field-Programmable Logic and Applications*; Brebner, G., Woods, R., Eds.; Springer: Berlin/Heidelberg, Germany, 2001; pp. 37–47.
12. Balachandran, S.; Kannan, P.; Bhatia, D. On metrics for comparing interconnect estimation methods for FPGAs. *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.* **2004**, *12*, 381–385. [CrossRef]
13. Kannan, P.; Bhatia, D. Interconnect Estimation for FPGAs. *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst.* **2006**, *25*, 1523–1534. [CrossRef]
14. Maarouf, D.; Alhyari, A.; Abuowaimer, Z.; Martin, T.; Gunter, A.; Grewal, G.; Areibi, S.; Vannelli, A. Machine-Learning Based Congestion Estimation for Modern FPGAs. In Proceedings of the 2018 28th International Conference on Field Programmable Logic and Applications (FPL), Dublin, Ireland, 27–31 August 2018; pp. 427–4277. [CrossRef]
15. Pui, C.; Chen, G.; Ma, Y.; Young, E.F.Y.; Yu, B. Clock-aware ultrascale FPGA placement with machine learning routability prediction: (Invited paper). In Proceedings of the 2017 IEEE/ACM International Conference on Computer-Aided Design (ICCAD), Irvine, CA, USA, 13–16 November 2017; pp. 929–936. [CrossRef]
16. Vivado Development Team. Xilinx Vivado. Available online: https://www.xilinx.com/products/design-tools/vivado.html (accessed on 17 August 2021).
17. Yang, S.; Gayasen, A.; Mulpuri, C.; Reddy, S.; Aggarwal, R. Routability-Driven FPGA Placement Contest. In Proceedings of the 2016 on International Symposium on Physical Design, Santa Rosa, CA, USA, 3–6 April 2016; ACM: New York, NY, USA, 2016; pp. 139–143. [CrossRef]
18. Balachandran, S.; Bhatia, D. A-priori Wirelength and Interconnect Estimation based on Circuit Characteristics. *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst.* **2005**, *24*, 1054–1065. [CrossRef]
19. Zhao, J.; Liang, T.; Sinha, S.; Zhang, W. Machine Learning Based Routing Congestion Prediction in FPGA High-Level Synthesis. In Proceedings of the 2019 Design, Automation Test in Europe Conference Exhibition (DATE), Florence, Italy, 25–29 March 2019; pp. 1130–1135. [CrossRef]
20. Xilinx Ultrascale+ Product Details. Available online: https://www.xilinx.com/support/documentation/product-briefs/virtex-ultrascale-plus-product-brief.pdf (accessed on 17 August 2021).
21. Large FPGA Methodology Guide, Including Stacked Silicon Interconnect (SSI) Technology. Available online: https://www.xilinx.com/support/documentation/sw_manuals/xilinx14_7/ug872_largefpga.pdf (accessed on 17 August 2021).
22. Xilinx Architecture Terminology. Available online: https://www.rapidwright.io/docs/Xilinx_Architecture.html (accessed on 17 August 2021).
23. Comet ML Supercharging Machine Learning Tool. Available online: www.comet.ml (accessed on 17 August 2021).
24. ELI5 Tool. Available online: https://eli5.readthedocs.io/en/latest/ (accessed on 17 August 2021).
25. Amazon SageMaker Machine Learning for Every Data Scientist and Develope. Available online: https://aws.amazon.com/sagemaker/l (accessed on 17 August 2021).
26. Friedman, J.H. Multivariate Adaptive Regression Splines. *Ann. Statist.* **1991**, *19*, 1–67. [CrossRef]
27. Al-Hyari, A.; Abuowaimer, Z.; Martin, T.; Gréwal, G.; Areibi, S.; Vannelli, A. Novel Congestion-Estimation and Routability-Prediction Methods Based on Machine Learning for Modern FPGAs. *ACM Trans. Reconfigurable Technol. Syst.* **2019**, *12*. [CrossRef]
28. Data Repository for Machine Learning Based Congestion Estimation. Available online: https://personal.utdallas.edu/~dinesh/Data/ (accessed on 17 August 2021).