BEYKOZ UNIVERSITY

GRADUATE PROGRAMS INSTITUTE

APPLYING ARTIFICIAL INTELLIGENCE FOR FPGA PHYSICAL DESIGN AUTOMATION TOOLS

Cheikhsaadbouh ETFAGHAOUBEID

MASTER'S THESIS

İstanbul, November 2024

APPLYING ARTIFICIAL INTELLIGENCE FOR FPGA PHYSICAL DESIGN AUTOMATION TOOLS

Cheikhsaadbouh ETFAGHAOUBEID

MASTER'S THESIS
COMPUTER ENGINEERING DEPARTMENT
COMPUTER ENGINEERING MASTER'S PROGRAM WITH THESIS

MASTER THESIS ADVISOR

Asst. Prof. Dr. Özlem Feyza ERKAN

MASTER'S THESIS JURY MEMBERS

Asst. Prof. Dr. İnal Begüm TURNA DEMİREL

Asst. Prof. Dr. Halime SUVAY EKER

PREFACE

I would like to express my sincere gratitude to my mentors and advisors, Dr. Özlem Feyza Erkan and Dr. Shahzad Ahmed Memon, for their continuous support throughout my Master's study and related research. Their patience, motivation, and immense knowledge were invaluable. Their guidance helped me immensely during all stages of research and writing this thesis. I could not have imagined having better advisors and mentors for my Master's study.

I would also like to thank Dr. Yehdhih Mohamed Moctar from Lattice Semiconductor for his insightful comments, encouragement, and challenging questions that pushed me to broaden my perspective from various angles.

Finally, I would like to thank my family for their unwavering support throughout my Master's journey and life in general. Your prayers for me have sustained me thus far.

CONTENTS

ÖZET

FPGA FİZİKSEL TASARIM OTOMASYONU İÇİN YAPAY ZEKANIN UYGULANMASI

Yerleştirme aşamasında FPGA tasarımlarında yönlendirme tıkanıklığını tahmin etmek için yeni bir çerçeve sağlıyoruz. Giriş net listesini düzenlemek için yük dengeli bir iki bölümleme stratejisi kullanan yaklaşımımız, yönlendirme bilgisi tahmininin doğruluğunu artırır. Çerçeve, yerleştirme şemasını ve tasarım bağlantısını yansıtan özenle seçilmiş özellikleri içerir. Giriş ağ listesine yönelik bu kümeleme yaklaşımı yalnızca mevcut metodolojileri geliştirmekle kalmaz, aynı zamanda modelin yönlendirme bilgilerini doğru bir şekilde tahmin etme yeteneğini de önemli ölçüde geliştirir. Ek olarak bu strateji, oldukça sıkışık tasarımların yönlendirilebilirliğini artırır ve yönlendirici çalışma süresini önemli ölçüde azaltır.

Yöntemimiz, önemli ölçüde daha hızlı bir çalışma zamanında çalışırken, ilk yönlendirmeyle karşılaştırılabilir yönlendirilebilirlik tahmin doğruluğu elde eder ve genel tasarım yönlendirilebilirlik tahminlerinde hız ve hassasiyetin ikili zorluğunu ele alır. Giriş görüntüsünün farklı kanallarına kodlanmış iyi tasarlanmış özelliklerin girişini sağlayan önerilen çerçeve, mevcut yöntemlere göre üstün performans göstermektedir. Bu başarı, giriş ağ listesinin bağlantı ve sınırlayıcı kutuya dayalı olarak etkili bir şekilde bölümlenmesine atfedilir.

Erken aşamadaki FPGA yönlendirme mimarisi araştırmasında modelimiz, hızlı ve doğru tasarım yinelemelerine olanak sağlayarak değerli olduğunu kanıtlıyor. Ulaşılabilir maksimum frekans (Fmax), minimum kanal genişliği (Wmin) ve en kötü gevşek gecikme gibi önemli yönlendirme ölçümlerini doğru şekilde tahmin ederek tam VPR CAD akışı sonuçlarıyla güçlü bir korelasyon gösterir, ancak daha hızlıdır.

Anahtar Kelimeler: FPGA, Yapay Zeka, Fiziksel tasarım, Tıkanıklık, DRC ihlali, Yönlendirme bilgisi.

Tarih: 01/11/2024

ABSTRACT

APPLYING ARTIFICIAL INTELLIGENCE FOR FPGA PHYSICAL DESIGN AUTOMATION TOOLS

We introduce a novel framework for predicting routing congestion in FPGA designs during the placement phase. By employing a load-balanced bi-partitioning strategy to organize the input net list, our approach enhances the accuracy of routing information prediction. Our framework incorporates carefully selected features that reflect the placement configuration and design connectivity. This clustering approach to the input net list not only improves current methodologies but also significantly improves the model's ability to predict routing congestion accurately. Additionally, this strategy enhances the routability of highly congested designs and substantially reduces router runtime.

Our method achieves routability forecasting accuracy comparable to initial routing while operating at a significantly faster runtime, this address the critical need for both speed and precision in general design routability predictions. The proposed framework, which inputs well-engineered features encoded on across multiple input picture channels, shows improved performance over current techniques. This improvement is attributed to the effective partitioning of the input net list based on connectivity and bounding box.

Our model is a valuable tool for early-stage FPGA routing architecture exploration, enabling rapid and accurate design iterations. It accurately predicts key routing metrics such as maximum achievable frequency (Fmax), minimum channel width (*Wmin*), and worst slack delay, the models' predictions exhibit correlation with full VPR CAD flow results but at a faster speed.

Keywords: FPGA, AI, Physical design, Congestion, DRC violation, Routing information.
Date: 01/11/2024

ABBREVIATIONS

AI: Artificial Intelligence

CAD: Computer-Aided Design

CLB: Configurable Logic Block

CNN: Convolutional Neural Network

DL: Deep Learning

DRCs: Design Rule Checks

FPGA: Field Programmable Gate Array

FFs: Flip-Flops

HDL: Hardware Description Language

LUTs: Look Up Tables

MAE: Mean Absolute Error

ML: Machine Learning

NN: Neural Network

NP-hard: Non-deterministic Polynomial-time hard

I/O: Input and Output

VPR: Versatile Placement & Route

VTR: Verilog To Routing

LIST OF FIGURES

LIST OF TABLES

1 INTRODUCTION

About 25 years ago, Field Programmable Gate Arrays (FPGAs) were initially made available. Since then, they have experienced remarkable expansion and have gained popularity as a medium for implementing digital circuits. Process technology has advanced significantly, increasing FPGA's logic capacity and making them feasible implementation option for more intricate and large-scale designs. However, the space, speed and power consumption of the final device are significantly impacted by the configurable nature of their logic and routing resources. Additionally, with all these gained still have some problem specially in physical design like routing congestion and timing.

Over the past two decades, machine learning has drawn more attention for a wide range of applications, including Computer Aided Design (CAD). The primary appeal of machine learning (ML) lies in its ability to learn from data, adapt to new tasks and give some input. This eliminates the need for hardcoding and enables the development of more flexible and adaptable systems. Moreover, ML is highly generalizable to novel problems. Deep learning (DL), a subset of machine learning, has demonstrated exceptional capabilities in learning complex mappings from input data.

By utilizing cutting-edge ML and DL models to address the issues and constraints of the present FPGA placement and routing phases process, this thesis seeks to achieve a more intelligent FPGA placement and routing flow.

1.1 MOTIVATION

The work presented in this thesis is motivated by the numerous challenges associated with FPGA implementation, including:

- The Complexity of FPGA Designs: Modern FPGAs have surpassed the 30 billion transistor integration level, making them large and more complicated. Fitting designs onto an FPGA while considering resource constraints and achieving timing closure is a significant challenge. Additionally, applications targeting FPGAs are becoming more complex, further complicating placement and routing tasks.

- FPGA Placement and Routing Runtime: For large and complex designs, FPGA compilation times can extend to hours or even days, even with heuristics that compromise the best solution for a faster runtime. This significantly slows down the FPGA development cycle.
- Sub-Optimal Solutions: Many FPGA placement and routing optimization problems are classified as NP-hard. Heuristics used in current CAD tools address these challenges by sacrificing computational efficiency for optimal solutions, aiming to find workable solutions quickly.
- Limited application of Machine Learning: While there has been a significant amount of work in using machine learning techniques to estimate or predict design routability, to best of our knowledge, none of this work has attempted to employ clustering or partitioning techniques on the input net list to improve ML model predictability.

To address these challenges, this thesis introduces a load-balanced partitioning approach to cluster the input netlist into partitions within the same bounding box on the FPGA. This approach aims to improve the accuracy and efficiency of machine learning models used in FPGA placement and routing.

1.2 CONTRIBUTIONS

This thesis makes three main contributions to the use of machine learning techniques in improving FPGA CAD physical design:

- Introduced a Load-Balanced Partitioning Approach: Applied to the input net list, this approach enhances feature extraction and routing information prediction. It facilitates fine-tuning these predictions, providing guidance for the routing algorithm throughout the iterative process, thereby improving the routability of congested designs.
- Predicted Congested Areas: Utilized extracted features to forecast congested regions within each partition of the FPGA using a transfer learning approach. This approach leverages existing pretrained machine learning models from TensorFlow Hub to train the model (*TensorFlow Hub*, n.d.).

.

- Predicted Routing Information: Developed a random forest regression model and trained it to estimate key routing details such as the number of design rule violations, minimum channel width, and the maximum achievable frequency (F_{max}) of the design. These predictions were then used to guide the router in successfully routing a real-world design that previously failed due to congestion.

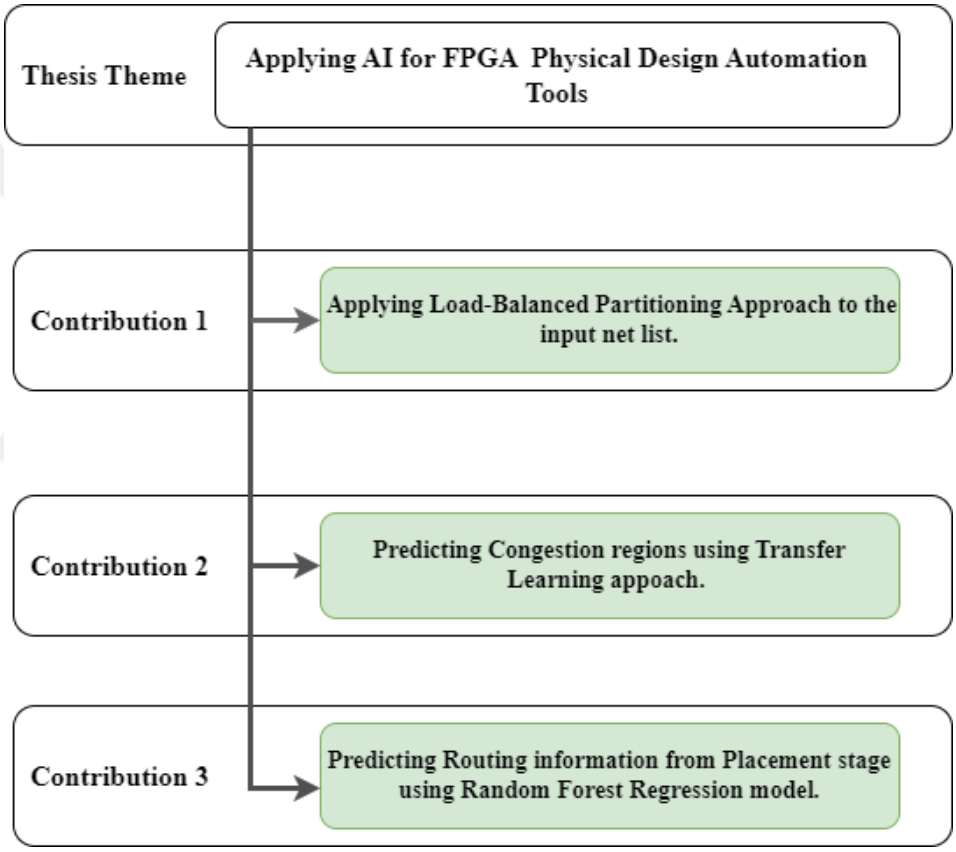

Figure 1.1: Thesis Contributions

1.3 THESIS ORGANISATION

In the background chapter, we present necessary information on FPGAs and the placement stage of the FPGA CAD flow. Additionally, it provides details on DL, ML, and the architecture employed in this work. The literature review encompasses research conducted on the routability prediction, congestion management, congestion estimation, and

the use of machine learning in these areas. In the Methodology chapter, we propose a new approach for addressing FPGA design challenges using deep learning algorithms and estimate the expected improvements. In the Experiments and Results chapter, we present the results of our work. Finally, the last chapter summarizes our findings and discuss potential future directions.

2 BACKGROUND

This chapter provides the background information necessary for the following chapters. Section 2.1 explains the stages involved in the FCAD flow. In Section 2.2, we will explore placement and routing in more detail to understand this phase within the CAD flow. We will also discuss the challenges encountered in this section. Section 2.3 covers the fundamentals of machine learning and deep learning. The machine learning and deep learning models that have been employed in this study are also covered in Section 2.3.5.

2.1 FPGA CAD FLOW

The process of employing specialized software tools to design and construct an FPGA is referred to as the FPGA design flow. Designed to be highly adaptable and suitable for a wide range of applications, FPGAs are integrated circuits that can be customized to execute specific tasks after manufacturing (Bhardwaj, n.d.).

- **Register Transfer Level (RTL) Design:** RTL is an abstraction of a combinational and sequential digital circuit. RTL is described in a Hardware Description Language (HDL) such as Verilog or VHDL. RTL designers often build RTL for their sub-systems, focusing on functionality and leaving the optimization of power and performance to physical design tools.
- **Synthesis:** Synthesis is the process of translating an RTL description into a gate-level netlist. The flip-flops and logic gates in this netlist serve as representations of the design.
- **Optimization:** Optimization focuses on meeting various constraints, such as timing constraints (guaranteeing correct frequency), area constraints (restricting resource utilization), and power constraints (controlling power consumption).
- **Placement and Routing:** In this stage, the generated netlist is transferred into the FPGA device's physical resources. Placement tools determine the location of each logic element within the device, while routing tools create the connections between these elements using the available routing resources (interconnects). We will discuss placement and routing in more detail later.

- **Timing Analysis:** After placement and routing, timing analysis verifies that the design satisfies its timing constraints. Timing analysis tools ensure that signal propagation delays along the logic circuits do not violate setup and hold time requirements.
- **Bitstream Generation:** If the design passes timing analysis, the latest stage is to generate the bitstream, which is the binary file that sets up the FPGA device to carry out the required functionality. Configuration information for the programmable logic, I/O, and other resources of the FPGA is included in the bitstream.

After generating the bitstream, the next step is verification and testing. While not strictly part of the CAD flow, it is crucial to ensure the design functions correctly. Engineers install the device's firmware onto the FPGA and perform functional testing and verification. To ensure the design meets all functional requirements, it must be rigorously tested using a variety of test vectors.

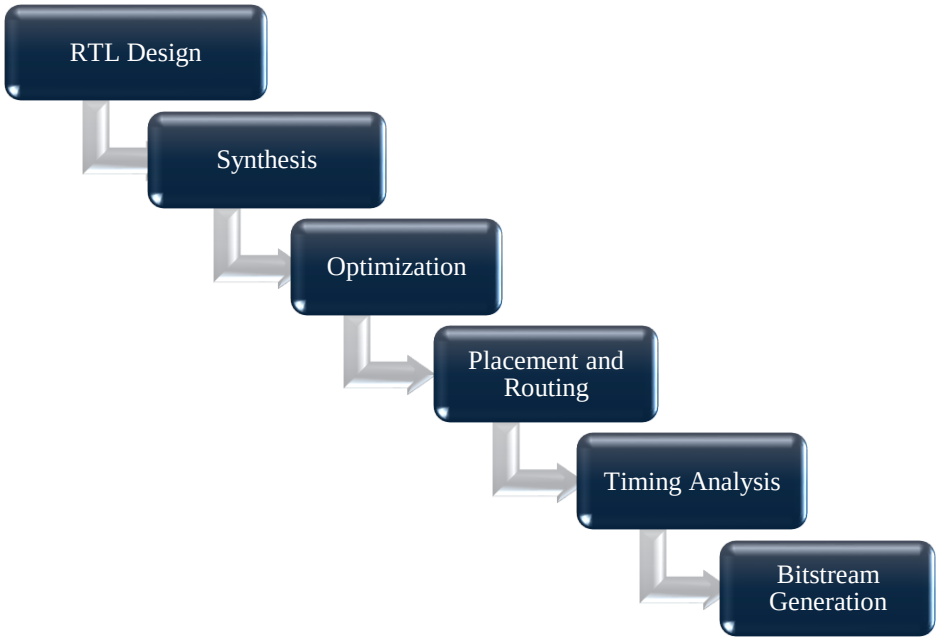

Figure 2.1: FPGA CAD Flowchart

That was an overview of the FPGA CAD flow, providing a general representation of all the steps involved. We will not delve too deeply into each phase, as our primary focus is on the placement and routing phase. This is the most complex phase in the entire CAD flow, and we aim to enhance it using machine learning techniques. We will discuss this phase in detail later in the thesis.

2.2 PLACEMENT AND ROUTING

In the physical design of Field Programmable Gate Arrays (FPGAs) and integrated circuits (ICs), placement and routing (P&R) are essential phases. Placement is the process of determining the optimal locations for circuit components (logic gates, memory blocks, etc.) on a chip to minimize space, power consumption, and delay while adhering to design specifications. Routing, which follows placement, is the process of connecting these placed components using metal interconnects. The goal is to ensure that signals can be routed effectively between components without exceeding timing constraints, creating congestion, or interfering with other signals.

2.2.1 Placement

Placement algorithms determine which FPGA logic block (instance) corresponds to the matching logic block required by the circuit. To reduce wiring length, connected logic blocks are placed close to one another (wire length-driven layout). Occasionally, blocks are placed to balance the wire density across the FPGA (routability-driven placement) or to maximize circuit speed (timing-driven placement) (Farooq et al., 2012). Placers can be categorized into three main types: min-cut (partitioning-based), simulated annealing-based, and analytical placers, often followed by local iterative improvement. To ensure fair evaluation of designs, it is essential that every feature of the FPGA is utilized effectively by our CAD tools. This implies that the placer's optimization strategy and objectives may vary depending on the specific architecture. The two most popular methods used in FPGA CAD tools are partitioning-based and simulated annealing-based approaches (Farooq et al., 2012). Therefore, we will concentrate on these two strategies in the following part.

- **Simulated Annealing Based Approach**

One technique that simulates the annealing process, which gradually cools molten metal gradually to produce high-quality metal products, is called "simulated annealing." The pseudo-code for a generic placer based on simulated annealing is shown in Algorithm 2.1. A cost function is used to evaluate the quality of a particular logic block placement. For example, in wire-length-driven placement, the sum of the half-perimeters of the bounding boxes of all nets is a common cost function. Initially, the available spots on the FPGA are randomly assigned to logic blocks. Subsequently, the placement undergoes several local and minor adjustments to progressively improve it. A logic block and its new position are selected randomly.

Algorithm 2.1: Generic Simulated Annealing-based Placer Algorithm

1. S = RandomPlacement ();
2. T = InitialTemperature ();
3. R_{limit} = $Initial_{limit}$;
4. **while** ExitCriterion () == false **do**
5. **while** InnerLoopCriterion() == false **do**
6. S_{new} = GenerateViaMove (S, R_{limit});
7. ΔC = Cost (S_{new}) - Cost(S);
8. r = random (0,1);
9. **if** r < $e^{-\Delta C/T}$ **then**
10. S = S_{new}
11. **end if**
12. **end while**
13. T = update Temperature ();
14. R_{limit} = update R_{limit}();
15. **end while**

where T is a temperature parameter that regulates the likelihood of accepting changes that degrade placement, while ΔC represents the change in the cost function. Initially, T is set high enough to accept almost any move. As the placement improves, T is gradually

lowered, reducing the probability of accepting moves that worsen the solution. This allows simulated annealing to escape local minima by accepting occasional uphill moves.

The R_{limit} parameter determines the maximum distance between blocks that can be swapped. Initially, R_{limit} is set relatively high to allow for long-range swaps on a chip. As the annealing process progresses, R_{limit} is adjusted to maintain a balance between exploration and exploitation. If the acceptance rate, α, exceeds 0.44, R_{limit} is increased to encourage more exploration. Conversely, if α, is less than 0.44, R_{limit} is decreased to focus on local optimization (D. Chen et al., 2006).

The total wirelength of the current placement determines the objective cost function. An estimate of the routing resources required to fully route each net in the netlist is provided by the wirelength. As wirelength decreases, fewer switches and routing tracks are needed to route nets. Given the limited routing resources of an FPGA, this is a crucial factor to consider. Fewer switches and routing tracks generally lead to shorter routing net delays between logic units. Equation 2.1 calculates the total wirelength of a placement using a semi-perimeter measure. *N* represents the total number of nets; and *bbx(i)*, *bby(i)*, and *q(i)* represent the horizontal and vertical spans and a correction factor for each net, respectively. Figure 2.2 illustrates the calculation of the horizontal and vertical spans of an imaginary six-terminal system.

$$WireCost = \sum_{i=1}^{N} q(i) \times \left(bbx(i) + bby(i)\right) \quad (2.1)$$

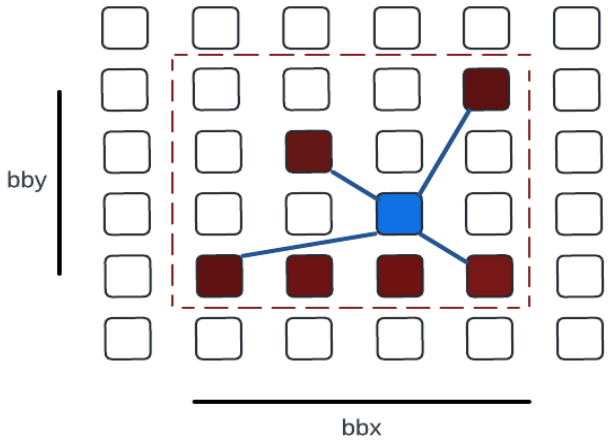

Figure 2.2: Boundary Box of an Imaginary Six-Terminal Net

The annealing schedule specifies the number of attempted moves at each temperature (InnerLoopCriterion), the criteria for terminating the annealing process, the rate at which the temperature decreases, and the method for generating potential moves. To achieve satisfactory results in a reasonable amount of CPU time, an effective annealing schedule is essential. Many proposed annealing schedules are “fixed” programs that cannot be adapted to fit various issues. These schedules lack flexibility and are not suitable for a wide range of applications, but they can be effective within their specific design space (D. Chen et al., 2006).

- **Partitioning Based Approach**

Partitioning-based placement strategies are based on graph partitioning algorithms, such as the Fiduccia-Mattheyses (FM) and Kernighan-Lin (KL) algorithms. Tree-based FPGA designs are well-suited for placement based on partitioning. The partitioner is used to repeatedly divide netlist cells into clusters at each hierarchical level. Reducing outgoing communications and consolidating densely connected cells into a unified cluster are the goals.

Mesh-based FPGA also uses partitioning-based placement. The device is split into two halves, with one partition containing the densely connected blocks. A circuit partitioning approach is used to determine which partition each logic block belongs to, with the goal of minimizing the number of cuts in the nets linking the blocks across partitions (D. Chen et al., 2006).

2.2.2 Routing

As we have stated, routing is an essential phase in the FPGA CAD Flow. The core challenge in FPGA routing lies in assigning nets to routing resources without any resource being shared by multiple nets. Currently, Pathfinder stands as one of the most advanced FPGA routing technique available. Pathfinder makes advantage of the directed graph abstraction G(*V*,*E*) of the FPGA's routing resources. The set of vertices *V* in the graph represents the I/O terminals of logic blocks and the routing wires in the interconnect structure. A possible path connecting two vertices is called an edge. Figure 2.3 shows a portion of the routing graph for a mesh-based interconnect.

The routing problem for a given net in this graph model is to find an embedded directed tree in G that connects each of its sink terminals to the net's source terminal. The limited routing resources of an FPGA can make it challenging to design unique, non-intersecting trees for each net in a netlist.

Pathfinder routes each net in a netlist effectively by utilizing an iterative negotiation-based method. During the initial routing iteration, nets are routed individually using Dijkstra's shortest path algorithm without considering resource sharing This can lead to resource congestion, as multiple nets may compete for the same resources. To address this, subsequent iterations incorporate a resource congestion cost. This cost is determined by the resource's historical congestion history and the number of nets currently sharing it. Because of this, nets are made to haggle over routing resources. When a resource is very busy, nets that can use alternatives with less congestion are forced to do so. However, if a net has no viable alternatives, it may still be forced to use a congested resource. Equation 2.2 calculates the cost of utilizing a routing resource *n* during a routing iteration.

$$C_n = (b_n + h_n) * p_n \tag{2.2}$$

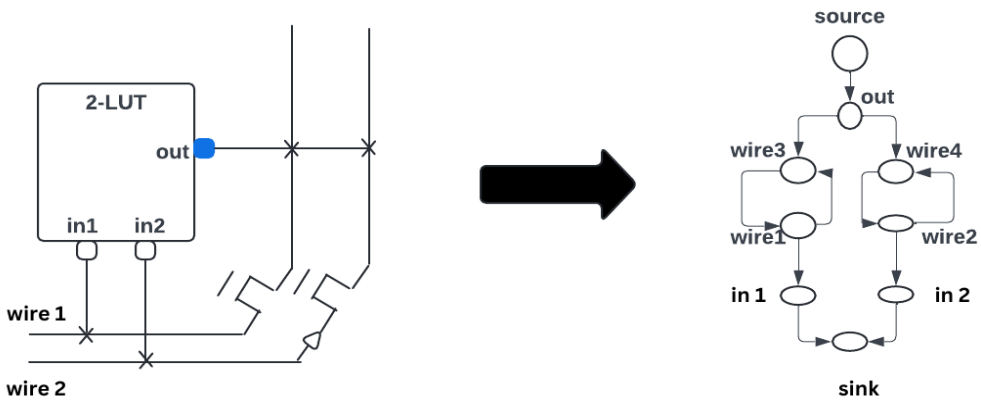

Figure 2.3: FPGA Architecture Represented as a Directed Graph

The base cost (b_n) of using resource *n* is determined its historical congestion *(h_n)* in previous iterations. The current congestion, represented by the number of nets sharing the resource (p_n), is proportional to historical congestion. The cost of utilizing a shared resource

n is represented by the h_n term, while p_n accounts for the current demand on the resource. The underlying principle is that a historically congested node should appear expensive, even if it is currently lightly loaded. This strategy helps to balance the load across the routing resources. Algorithm 2.2 presents the pseudocode for the Pathfinder routing algorithm.

Algorithm 2.2: Pathfinder Pseudo-code Routing Algorithm

```
1.  Let: RT_i be the set of nodes in the current routing of net i
2.  while shared resources exist do
3.      /*Illegal routing /*
4.      foreach net, i do
5.          rip-up routing tree RT_i;
6.          RT(i) = S_i;
7.          foreach sink T_ij do
8.              Initialize priority queue PQ to RT_i at cost();
9.              while sink T_ij not found do
10.                 Remove lowest cost node m from PQ;
11.                 foreach fanout node n of node m do
12.                     Add n to PQ at PathCost(n) = C_n + PathCost(m)
13.                 end foreach
14.             end while
15.             foreach node n in path T_ij to S_i do
16.                 /*backtrace*/
17.                 Update C_n;
18.                 Add n to RT_i;
19.             end foreach
20.         end foreach
21.     end foreach
22.     Update h_n for all n;
23. end while
```

The critical path delay is a crucial indicator of the routing quality generated by an FPGA routing algorithm. The critical path delay in a routed netlist is the longest delay among all combinational paths. A longer critical path delay results in a lower maximum operating frequency of the netlist. Pathfinder incorporates delay information into the revised resource cost Equation 2.3.

$$C_n = A_{ij} \times d_n + \left(1 - A_{ij}\right) \times (b_n + h_n) \times p_n \tag{2.3}$$

where d_n is the delay of the resource and A_{ij} is the criticality factor defined in Equation 2.4.

$$A_{ij} = D_{ij} / D_{max} \tag{2.4}$$

D_{max} is the critical delay of the netlist, and D_{ij} is the maximum delay of each combinational path that goes through the source and sink terminals of the net being routed. Equation 2.4 combines these two cost components. Pathfinder is one of the most advanced FPGA routing algorithms available today. Its negotiation-based approach effectively routes signals on FPGAs by balancing congestion and latency. Moreover, Pathfinder is a flexible routing method that can be represented as a directed routing graph. This is because it operates on a directed graph abstraction of the FPGA's routing structure.

2.3 AN OVERVIEW IN MACHINE LEARNING AND DEEP LEARNING

ML is a subset of Artificial Intelligence (AI). The goal of AI is to develop statistical models and algorithms that enable computers to learn from data and make decisions without requiring explicit programming for a specific activity. This involves training models using data to identify patterns and predict outcomes. A subset of machine learning called deep learning (DL) uses larger models, more data, and deeper architectures and layers. Figure 2.4 illustrates the relationship between AI, machine learning, and deep learning.

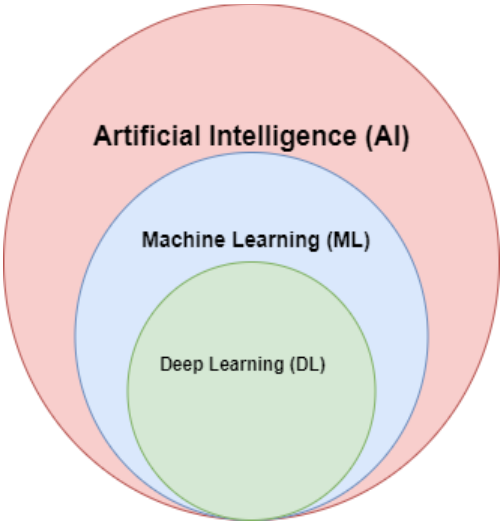

Figure 2.4: The Diagram of Artificial Intelligence and its Subsets

Machine learning can be divided into supervised learning, unsupervised learning and reinforcement learning.

2.3.1 Supervised Learning

Supervised learning is a machine learning method that relies on labeled data for training. Each training sample is associated with a known output value. The goal is to learn an input-to-output mapping that can be used to predict the labels of new, unseen data.

- **Classification:** Predicting a discrete label (e.g., spam vs. non-spam)
- **Regression:** Predicting a continuous value (e.g., house prices)

2.3.2 Unsupervised Learning

Unsupervised learning is a machine learning method that is based on unlabeled data for training models. The goal is to discover underlying patterns and structures within the data.

- **Clustering:** The process of grouping similar data points (e.g., consumer segmentation)
- **Dimensionality Reduction:** Reducing the number of features in a dataset without losing significant information (e.g., Principal Components Analysis).

2.3.3 Reinforcement Learning

An agent that has been trained to make decisions by carrying out certain tasks and receiving rewards or penalties serves as the foundation for the machine learning technique known as reinforcement learning. The goal is to find a policy that minimizes the cumulative reward over time.

- **Q-Learning:** A value-based approach used for determining the best action-selection strategy.

2.3.4 Deep Learning

Deep Learning (DL) is a subset of ML that utilizes larger models, more data, and deeper architectures. Artificial Neural Networks are used in deep learning to execute complex calculations on vast volumes of data. This type of machine learning draws inspiration from the structure and capabilities of the human brain. Artificial neural networks

consist of artificial neurons, or nodes, arranged in layers illustrated in Figure 2.5. These layers are typically organized into three main types:

- Input layer
- Hidden layer(s)
- The output layer

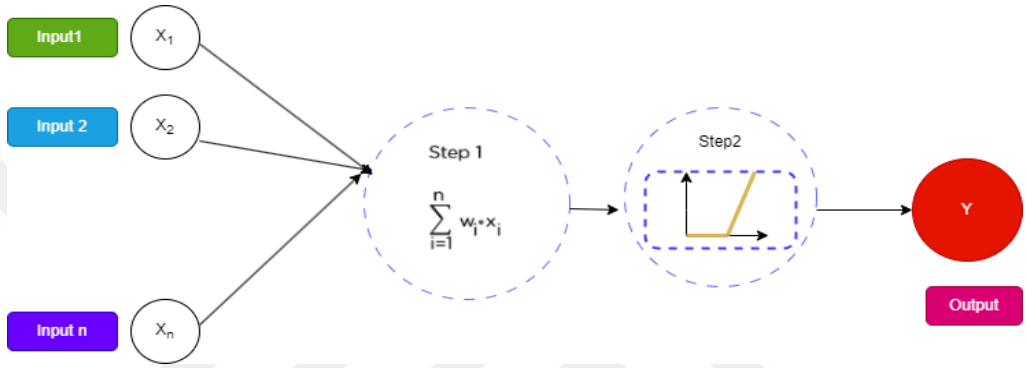

Figure 2.5: Artificial Neural Networks Process

Each node uses data inputs to obtain information. The node computes the inputs, multiplies them by random weights, and then adds a bias. Finally, a nonlinear activation function is applied to determine the output of the node.

Deep learning encompasses a vast toolbox of algorithms, each specialized for specific types of problems. While deep learning algorithms can handle a wide range of data types, they often require large amounts of data and computational power to address complex problems.

Below is widely used deep learning algorithms:

- Convolutional Neural Networks (CNNs)
- Long Short-Term Memory Networks (LSTMs)
- Recurrent Neural Networks (RNNs)
- Generative Adversarial Networks (GANs)
- Radial Basis Function Networks (RBFNs)
- Multilayer Perceptrons (MLPs)
- Self-Organizing Maps (SOMs)

- Deep Belief Networks (DBNs)
- Autoencoders

In this thesis, we will utilize Random Forest Regressor model and Convolutional Neural Network (CNN) model.

2.3.5 Machine Learning Models

In this work, we will focus on widely used machine learning methods for classification and regression problems. For classification, we will utilize transfer learning based on Convolutional Neural Networks (CNNs). For regression, we will employ three models (Linear Regression, Random Forest Regression and Decision Tree Regression) and compare their performance to identify the most suitable model for our specific regression problem.

2.3.5.1 Transfer Learning

Transfer learning is a machine learning technique in which a model training in one task is adapted in the second related task. Instead of building a model from scratch, transfer learning refers to using previously learned model as a starting point for a new task. Pre-trained models have the advantage of being trained on a large quantity of data, and often have optimal parameter values, unlike models trained from scratch. Transfer learning offers several benefits over training a model from scratch, including requiring less data, improving performance, reducing time and costs, and increasing adaptability.

These benefits are particularly useful in our case, where we have limited data. A pre-trained model has already learned features that can generalize well to similar tasks. Transfer learning is commonly used in domains like image classification and natural language processing (NLP) where large, pre-trained models (such as ResNet and BERT) are readily available.

2.3.5.2 Convolutional Neural Networks (CNNs)

CNNs, also called convNets, are muti-layered neural networks primarily used for image processing and object detection. CNNs have multiple layers that process and extract features from data:

- **Convolution Layer:** Extracts features from the input data using filters.

- **Rectified Linear Unit (ReLU):** ReLU is responsible for performing operations elements. The output is a rectified feature map.
- **Pooling Layer:** The rectified feature map is then passed to the pooling layer. Pooling involves downsampling the feature map to reduce its dimensionality. The resulting pooled feature map is flattened into a single, long continuous linear vector. This flattened vector is then fed into a fully connected layer for classification or regression. (Wang & Abadir, 2014).

The CNN process is depicted in Figure 2.6.

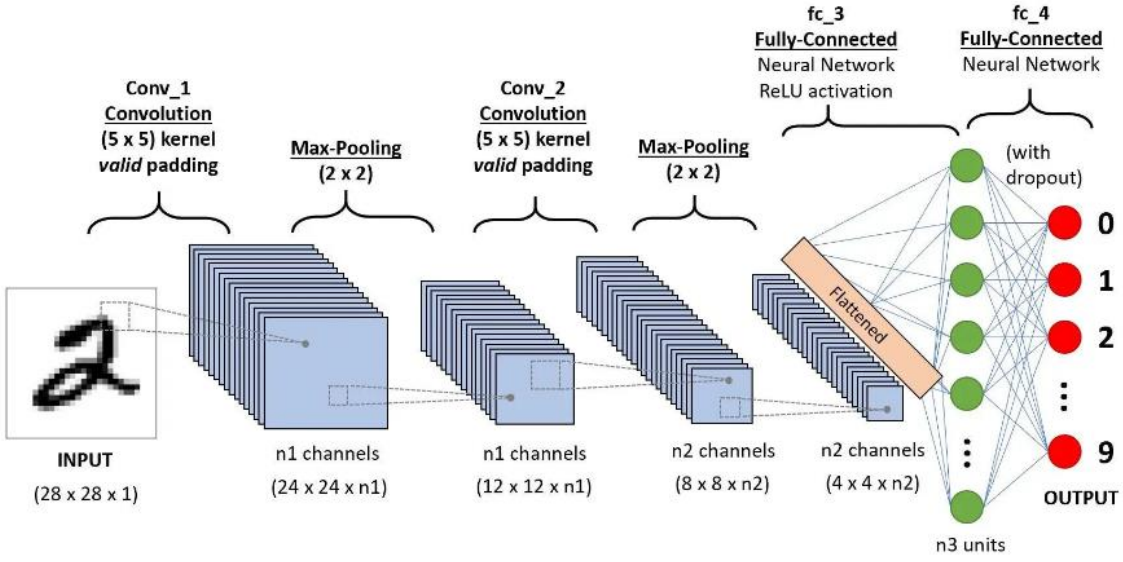

Figure 2.6: The Process of Convolutional Neural Network

2.3.5.3 Random Forest Regression Model

Random Forest is a supervised learning algorithm, that can only work with labeled data. It generates multiple decision trees, each trained on a random subset of the data. The model then creates an overall prediction for unknown data points by combining the results of each of these decision trees. The process is shown in Figure 2.7. Compared to individual decision trees, Random Forest can handle larger datasets and capture more complex relationships in the data.

Random Forest Regression is a type of supervised machine learning algorithm used for predicting continuous target variables. It employs an ensemble of decision trees. Bagging is used in the construction of each decision tree model separately. Smaller decision trees are

built from random subsets of the training data. The smaller models are combined to create the random forest model, which generates a single prediction value after the bagging phase. By merging the prediction from multiple decision trees, the algorithm reduces variance and improves accuracy.

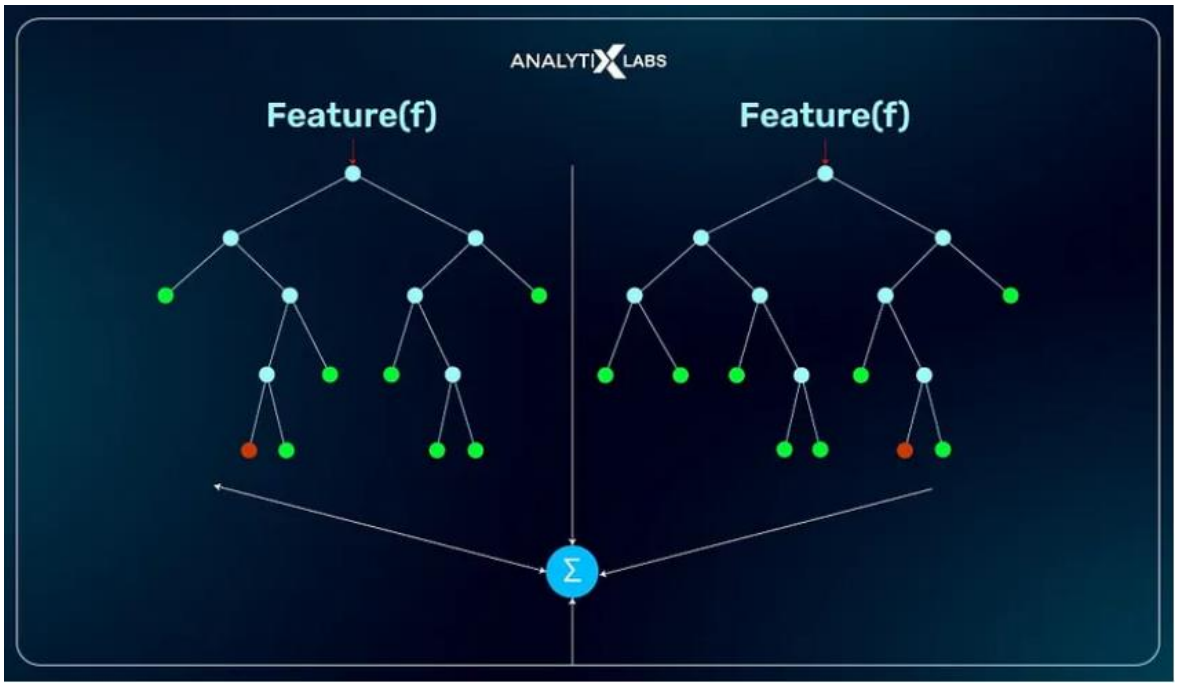

Figure 2.7: Random Forest Algorithm Process

There are many important parameters that influence the model's performance and accuracy, such as the number of trees (n_estimators), maximum depth of a tree (max_depth), minimum number of samples required to split a node (min_samples_split). We will experiment with different values for these parameters to optimize our model's performance.

2.3.5.4 Linear Regression Model

Linear regression is a type of supervised machine learning algorithm that models the linear relationship between a dependent variable and one or more independent variables by fitting a linear equation to observed data. There are two main types of liner regression: simple linear regression and multiple linear regression.

Finding the best fitting line is the primary goal when using linear regression, which minimizes the error between the predicted and actual values. The best-fitting line will have the least amount of error. The relationship between the dependent variable and independent variable is represented by a straight line in the best fit line equation. he slope of this line indicates how much the dependent variable changes for a unit change in the independent variable(s), as illustrated in Figure 2.8.

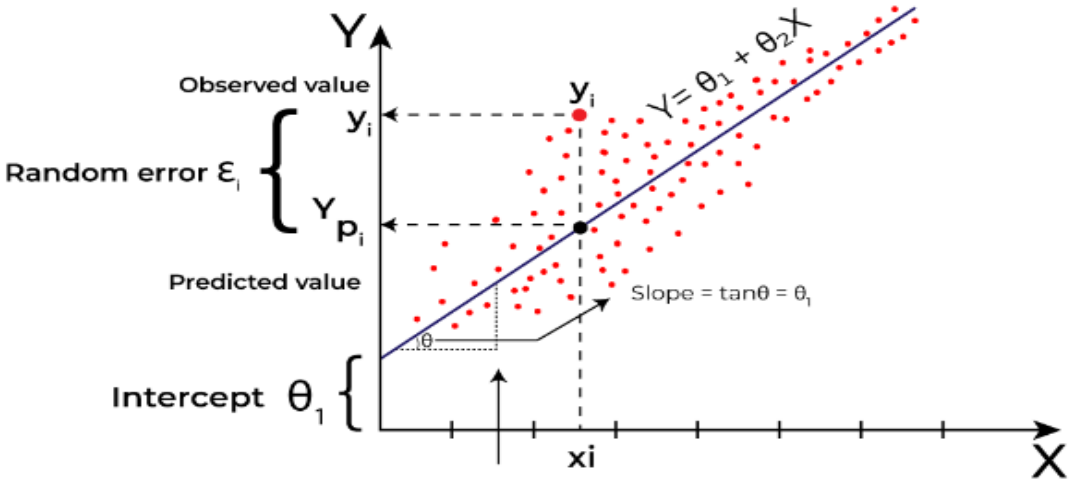

Figure 2.8: Linear Regression

Here, X is referred to as the independent variable or predictor variable, while Y is referred to as the dependent variable or target variable.

2.3.5.5 Decision Tree Regression Model

Decision Tree Regression is a machine learning technique that builds a tree-like model to predict continuous numerical values. It is more concerned with estimating numerical outcomes. A tree-like structure is the foundation of decision tree regression. Consider a dataset with various features and associated target values. The first step is to create a root node that represents the whole dataset.

The technique generates two child nodes, each of which represents a subset of the original data, after the initial split. The split condition determines the range of values each child node encompasses. The procedure is thereafter carried out by the algorithm for every

child node recursively dividing the data according to the best attributes and conditions. The Figure 2.9 illustrates this process.

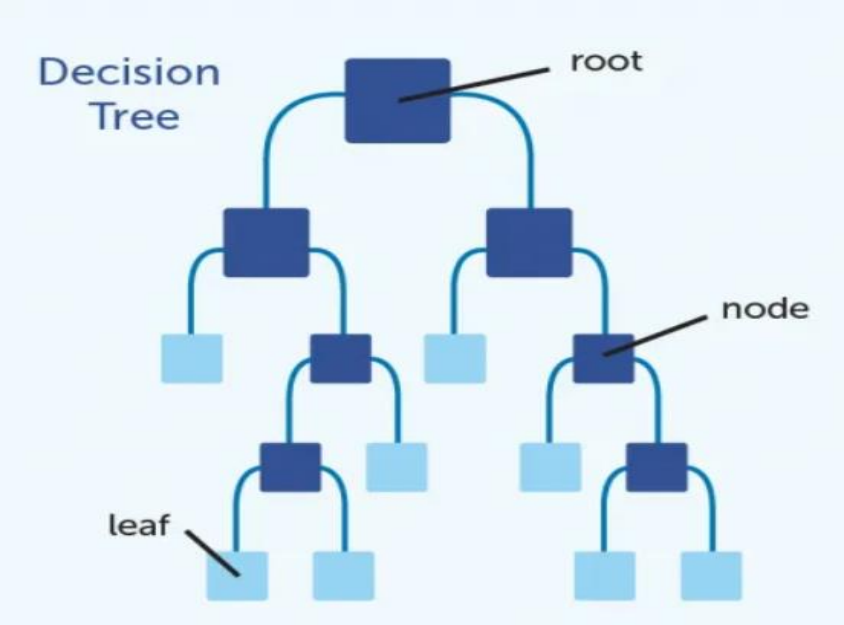

Figure 2.9: Decision Tree Process

2.3.6 Machine Learning Performance Measures

The ability to assess a model's performance is a crucial component of machine learning. For various machine learning tasks, there are numerous performance metrics, and each emphasizing a different aspect of the model's performance. A model may appear to perform well based on a single performance metric, but it might not perform well in other areas. Different performance metrics are used for classification and regression problems.

Measuring performance for classification models

- Accuracy measures the percentage of correctly predicted instances to the total

$$\frac{T_P + T_N}{(T_P + T_N + F_P + F_N)}$$

- Precision measures the proportion of the true positives out of all positive predictions

$$\frac{T_P}{(T_P + F_P)}$$

- Recall measures the proportion of the true positive of all actual positives

$$\frac{T_P}{(T_P + F_N)}$$

- F1 score the harmonic mean of the precision and recall

$$\frac{\text{Precision} \,.\, \text{Recall}}{(\text{Precision} + \text{Recall})}$$

where T_p: True Positive number, T_F: True Negative number, F_P: False Positive number, F_N: False Negative number.

Measuring performance for regression models

- Mean Absolute Error (MAE) measures the average magnitude of the errors without considering their direction.

$$\text{MAE} = \frac{1}{n}\sum_{n=1}^{n} |Y_i - \hat{Y}_i|$$

- Mean Squared Error (MSE) is the average squared difference between actual values and predicted values

$$\text{MSE} = \frac{1}{n}\sum_{n=1}^{n} (Y_i - \hat{Y}_i)^2$$

- Root Mean Squared Error measures the square root of the MSE

$$\text{RMSE} = \sqrt{\frac{1}{n}\sum_{n=1}^{n} (Y_i - \hat{Y}_i)^2}$$

- R-Squared (R^2) indicates how well the model explains the variance in the target variable

$$R^2 = 1 - \frac{\sum_{n=1}^{n} (Y_i - \hat{Y}_i)^2}{\sum_{n=1}^{n} (Y_i - \bar{Y})^2}$$

where Y_i is the actual value and $\hat{Y}_i$ is the predicted value for that instance.

3 LITERATURE REVIEW

This chapter introduces work on routing prediction from placement stage, as well as congestion estimation and management. We discuss both machine learning approaches and conventional heuristics employed in these tasks. The chapter is divided into three sections. A summary of relevant work in where machine learning used in CAD flow is presented in Section 3.1. Section 3.2 covers the tasks of routability estimation. In Section 3.3, we discuss congestion estimation in routing phase. Section 3.4 focuses on routability predictions techniques. Lastly, flow selection is provided in Section 3.5.

3.1 MACHINE LEARNING IN FPGA CAD FLOW

In the past several years, there has been a growing interest in applying machine learning to Electronic Design Automation (EDA) challenges. This is mainly because such models are easy to incorporate into existing EDA tools, are computationally efficient, and can learn and generalize, which results in notable increases in solution quality over conventional solution techniques. Wang and Abadir (2014) address the fundamental concepts of integrating machine learning into EDA and emphasize the primary challenges in feature creation for real-word applications. Kapre et al. (2015) developed an automated system named In-time, for selecting parameters that govern synthesis optimization. This reduces final timing results by 70% across industry benchmarks problem for the Altera CAD flow (Petelin, 2016).

To accelerate the search process, ML is utilized to predict the total wirelength of global interconnects in ASICs for a given Power Distribution Network (PDN) configuration (Al-Hyari, n.d.). A framework for efficient and high-quality Sub-Resolution Assist Feature creation using machine learning is presented in Xu et al., 2016. Mametjanov et al., (n.d.) propose a machine learning based technique for fine-tuning FPGA design parameters. The primary objective is to satisfy timing restrictions while remaining within 0.2% of the target power usage.

Deep learning is used by the authors in (Fuller, 2014) to correct divergence in timing endpoints across various Static Timing Analysis (STA) tools, including cell-arc delay, wire delay, stage delay, flip-flop setup time and path slack. Chan et al. (2017) propose a machine learning approach for Application Specific Integrated Circuits (ASICs) to predict Design Rule Check (DRC) violations after full routing. After that, the model prediction is included into a placement stage to improve optimization and reduce DRC violations. Overall, the machine learning model achieves a five-fold reduction in DRC violations and a 74% accuracy rate in predicting the locations of violating DRCs.

3.2 ROUTABILITY ESTIMATION

The routability estimation can be used to guide DRC violation mitigation techniques. In (Chang et al. 2024), the authors worked to develop a model architecture and select features for DRC hotspot identification in an efficient and effective manner. Their approach can provide a common set of raw features and effective model designs to aid in the development of routability prediction models.

Their main contribution is proposing to create routability estimators with minimal human intervention. They suggest an AutoML architecture that combines NAS (Network Attached Storage) with automatically matched feature selection. To the best of their knowledge, this is the first comprehensive AutoML framework for EDA. They propose a method for automatically selecting machine learning-based parameters that effectively and efficiently boots the model accuracy and search effectiveness of their NAS system. They introduce a graph propagation search technique to guide the search process towards high-quality designs and a graph-based search space that accommodates various actions and flexible connections. Their AutoML framework's model achieves a 9.9% improvement in the area under the receiver operational characteristics curve (ROC-AUC) compared to two typical routability estimators (J. Chen et al., 2020), (*ICCAD18_RouteNet.Pdf*, 2018). The fact that the full AutoML pipeline can be completed in a single day demonstrates the efficiency of their system.

3.3 ROUTING CONGESTION ESTIMATION

Since routing has become a significant problem due to limited FPGA resources fabric and the complexity of FPGA designs, numerous routing congestion prediction approaches have been presented in the literature specifically for FPGAs during the last 20 years. FGREP is a well-known congestion estimation technique created especially for FPG (Cheng et al., 2004).

Resource graph for routing was used by fGREP to provide a detailed congestion forecast. The complete range and constraints of available routing resources in FPGAs to the physical design are not fully understood, even though fGREP and fGREP2 often offer a reasonably accurate congestion estimation at the expense of a somewhat poorer runtime in compared to less precise techniques.

In (Elfadel et al., 2019) a ML based on a linear regression model was released to estimate congestion in routing stage. This model incorporates characteristics from neighboring sites as well as three features based on pin count and WLPA from each site. The model can estimate Vivado's congestion forecast once it has been trained, but it is not as good at predicting the real congestion that the Xilinx Vivado detailed router finds.

Zdun et al. (2023) propose a congestion map estimation for large-scale FPGA designs. They reframe the routing congestion problem as a high-definition image-to-image translation challenge, using placement stage characteristics and used a novel CGAN high-definition image-translation, namely pix2pixHD, to predict FPGA congestion. For large-scale designs, they suggest utilizing carefully selected features extracted from the placement stage, which can encode placement and connectivity information. When employing sophisticated image similarity assessment measures, their proposed methodology outperforms cutting-edge FPGA congestion map prediction in terms of modeling performance. By integrating their congestion prediction model into the placement engine, they were able to reduce routed wirelength by up to 7%.

Most congestion estimators ignore the wide variety of specific routing resource types that are available in FPGAs. For example, Dai (2003) takes into account all available routing resources in order to predict routability. However, FPGA routing resources can vary significantly in terms of length and connectivity. These can be as flexible as single length

lines, or they can be more constrained, like double and quad length lines thar are only appropriate for connections over greater distances.

A number of techniques for estimating routing congestion are provided in Coulson et al. (2020) to identify the congested areas for circuits before routing. Congestion estimates are first acquired using any of the several methods now in use, and they are subsequently processed as images. Subsequently, a technique for image blending is introduced, which allows for the iterative dissemination of congested areas within the image to adjacent areas with lower initial congestion. Additionally, a peak saturation approach that expands the most crowded locations is proposed for simulating the fixed channel widths. Findings indicate that congestion estimation techniques such as NCPR, those that rely on local rent exponents, and those that gauge channel use by running a single pass via a global router can all be enhanced by utilizing one or both of these methods. While these methods aid in increasing the accuracy of congestion, they do not take into account the characteristics of the FPGA routing architecture they are designed to simulate. Furthermore, they need to be experimentally adjusted for every congestion estimation technique. In the realm of machine learning, a method utilizing a linear regression model was recently introduced by Pui et al. (n.d.). This model uses three characteristics based on Wirelength Per-Area (WLPA) and pin count from each g-cell in the FPGA, along with one additional feature from nearby g-cells. After it has been trained, the model can accurately estimate the congestion forecast supplied by Vivado, but it is less accurate in predicting the actual congestion found by the detailed router.

3.4 ROUTABILITY PREDICTION TECHNIQUES

Routability prediction in the physical design is closely related to the interconnect congestion prediction problem and requires calculating the amount of routing resources needed before routing. Estimating congestion has taken on several shapes, with global routers being one of the most widely used (Grewal et al., 2017). But global routers have long runtimes, they can only be called seldom throughout placement, which reduces their overall usefulness. Furthermore, detailed routing reveals the often-significant implications of local nets on routability that global routers may overlook. Proposals have also been made for stochastic models (Lou et al., 2001). But most of the time, these models have limited

knowledge about connection geometry. Instead, they employ connection length probability distributions that are empirically obtained via the examination of serval circuit architectures. While heuristics like those proposed by Chan et al. (2017) can offer more precise congestion estimations, these instruments are still unable to account for the wide range of routing resources seen in heterogeneous FPGA devices.

Yang et al. (2016) emphasize the significance of taking routability into account early in the FPGA CAD flow when it comes to conventional, algorithmic routability solutions, since doing so results in quicker and higher-quality designs. They provide a routability-driven clustering technique, which embeds routability measurements into a cost function. Compared to VPack, their approach improves the minimum number of routing tracks by an average of 16.5%. By advocating for a closer integration between placement and routing, the work in Al-Hyari (n.d.) highlights the significance of routable placements. This thesis presents a unique approach that enables the possibility to incorporate the routing principles to the placement stage, ensuring that the placement is routability driven and that needed feasible routers exist for all nets in the system.

The subject of routing prediction in FPGAs using a hierarchical routing design is examined by Goswami and Bhatia (2021). The paper's primary contribution is in its capacity to forecast routability right after the technology mapping phase. Chai et al. (2022) concentrate on the challenge of determining if FPGA designs are routable prior to place-and-route. Their proposed estimation method considers FPGA's routing design. The results demonstrate that it is possible to accurately identify unroutable designs. In a work by (Al-Hyari, n.d.), it is provided a unique Boolean Satisfiability-based routing and routability estimation framework. While their approach can be implemented for small circuits, more improvements are required for big, contemporary designs.

In another work, they suggest using a fully convolutional network model to forecast hotspots for congestion (Liu et al., n.d.). By integrating the prediction model into a placement engine, they can obtain a 5% increase in routed wirelength when compared to many cutting-edge placers.

3.5 FLOW SELECTION

As discussed, machine learning (ML) holds significant promise for enhancing Electronic Design Automation (EDA) tools, particularly in physical design. We propose leveraging ML to predict routing congestion maps during the placement phase, as well as DRC violations and general routing information. By integrating ML into CAD tools, we aim to improve the efficiency and quality of placement and routing processes.

4 METHODOLOGY

In this chapter, we propose a new approach for addressing FPGA physical design challenges using machine learning algorithms and estimate the expected improvements. Section 4.1 covers Recursive Bi-partitioning, followed by feature selection in Section 4.2. In Section 4.3, our experiment design is presented. Data collection and analysis is discussed in Section 4.4. Sections 4.5, 4.6, and 4.7 delve into the prediction of DRC violations, congestion, and routing information, respectively. Finally, Section 4.8 discusses model selection.

Placement and routing are the most critical and time-consuming steps in translating a circuit design to an FPGA architecture. To get the ideal layout, this technique is usually iterative and requires several rounds of place and route. Traditionally, placement was done without taking routing information into account. However, routability is becoming a crucial placement factor in modern FPGA placement engines. A faster design closure time may be achieved by reducing the number of iterations required to accomplish the desired design using fundamental routability estimating methods.

Based on information from the placement stage, the FPGA routing congestion prediction task uses inter-stage to predict the overall routing congestion map of a design. This has a number of real-world applications, such as choosing placement strategies and making modifications to lessen routability problems. It can also be valuable in guiding the router with data like congestion maps, especially for large-scale designs. The model predicts routing information as an output after receiving a placement solution and a design netlist as inputs.

To enhance feature extraction and congestion prediction, our model employs a recursive spatial bi-partitioning approach. This method divides the FPGA's netlist by recursively splitting it into regions. Each region, represented by a node in the process, only contains nets that fit entirely within its boundaries. Due to the spatial independence of regions at each level of the tree, nets within one node can be analyzed independently of those in other nodes. Our model can accurately predict any attribute shared by all the regions since feature extraction is done for each division or area of the FPGA.

The ability to quickly determine whether a routing challenge is very difficult or impossible to solve is crucial for high-speed routing. In these situations, it is critical to let the user know that the desired outcome is either impossible to achieve or would take a long time. To predict these scenarios after deployment, we need to correlate routing difficulties with factors that can be readily retrieved from the target FPGA and the user circuit. W_{min} is defined as the minimal number of tracks per channel required for successful routing for a particular circuit with certain location. The directed search method may be used to swiftly solve the routing problem if the number of tracks available in the FPGA is 10-20% more than W_{min}. However, the difficulty of the challenge increases when the number of tracks per channel drops below this threshold.

In this thesis, we explore the use of ML to enhance FPGA CAD design in three key aspects:

- Provide guidance for the routing algorithm through the iterative process to minimize congestion and to better handle high fanout nets
- Select the best placement that can achieve better Quality of Results (QoR) and less congested routing
- Predict routing information such as minimum channel width, maximum achievable frequency (F_{max}) of the design

The next sections detail our load-balanced partitioning strategy for the input netlist, followed by a discussion of feature extraction at various points in the physical design phase. We also introduce our approach to hotspot identification and the prediction of design rule violations.

4.1 RECURSIVE BI-PARTITIONING

One key contribution of this work is that we organized the input net list into regions of independently placed nets based on a load-balanced spatial partitioning approach. This enabled us to predict the routing information for each region independently. An example of 8 partitions is shown in Figure 4.1.

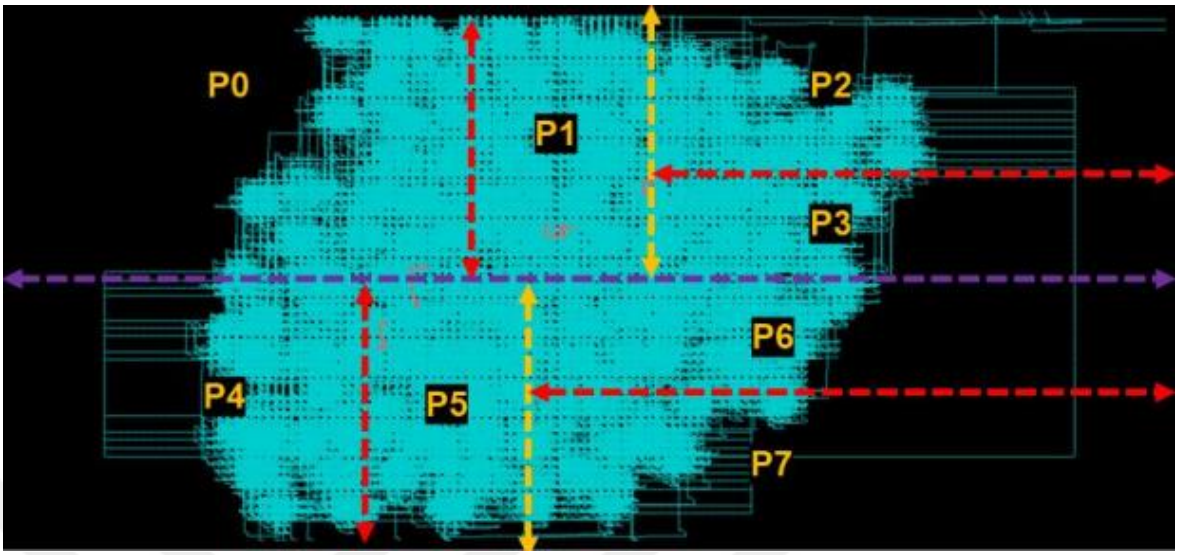

Figure 4.1: Actual Partitioning of a Net List using Our Load-Balanced Spatial Approach (Zhou et al., 2020)

By recursively bi-partitioning the FPGA, we create a perfect binary tree where each node represents a region on the FPGA that contains nets that fit entirely within its boundaries. A representative example of the bi-partitioning tree is illustrated in Figure 4.2. Since each node at a tree level reflects geographically distinct locations, nets inside a node may be routed independently of those of other nodes.

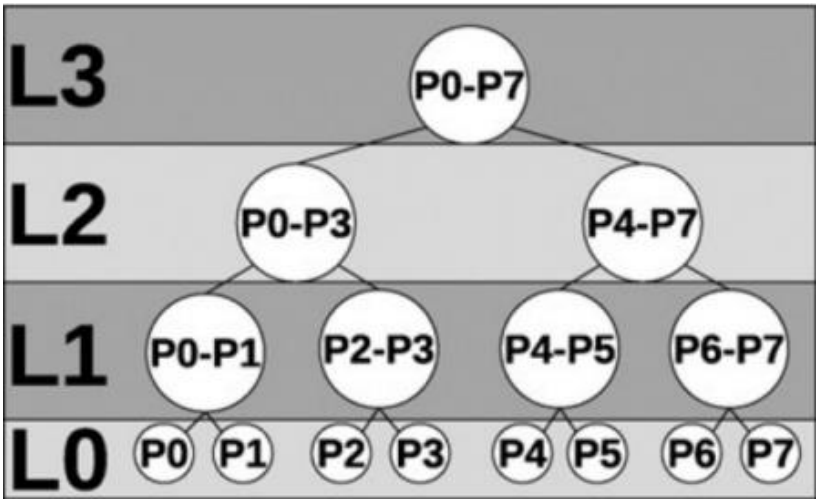

Figure 4.2: **An Illustration of a Four-Level Bi-Partitioning Tree (Zhou et al., 2020)**

We used an approach like that used in (J. Chen et al., 2020). Algorithm 4.1 shows the method starts by recursively dividing the FPGA into two distinct sections in order to minimize the difference in burden caused by routing the nets in the two regions.

Algorithm 4.1: The Recursive Bi-partitioning Algorithm

```
1.  procedure rec_Bipart (bbox, nets, number_of_levels, node)
2.      if current_level < number_of_levels - 1 then
3.          nets_before.nets_after.optimal_cut ← get_optimal_cut(bbox, nets)
4.          nets_crossing ← nets ← nets_before – nets_after
5.          node.nets ← nets_crossing
6.          bbox_before.bbox_right ← split_bbox(bbox, optimal_cut)
7.          rec_Bipart (bbox_before, nets_before, number_of_levels, current_level
        + 1, node.right)
8.          rec_Bipart (bbox_after, nets_after, number_of_levels, current_level + 1,
        node.right)
9.      else
10.         node.nets ← nets
11.     end if
12. end procedure
```

There are a limited number of cutlines to consider since the method uses the discrete nature of the FPGA architecture. While the discrete nature of FPGAs limits the number of possible bi-partitions (n rows + 1 horizontal cuts and n columns + 1 vertical cuts), employing a naive bi-partitioning technique can still be inefficient. For example, calculating the total workload of nets before and after each possible cutline to find the most load-balanced cutline results in an inefficient O(KN) strategy, where K is the number of cutlines and N is the number of nets. The bottleneck arises from repeatedly determining each net's position relative to every cutline.

Algorithm 4.1 details the steps involved in ParaDRo. The list of nets is iterated only once. During a single iteration over the netlist, the algorithm populates the nets_start, nets_end, total_workload_before, and total_workload_after arrays. Instead of finding the optimal cutline in O(KN) time, these arrays allow it to be discovered in (O + K) time.

Our model utilizes two data structures, *nets_start* and *nets_end*, to efficiently store information about nets and their bounding boxes. Each element in these arrays corresponds to an individual net. The element at index *i* in both arrays represents a net whose bounding box starts at the i-th row/column of the FPGA and ends at an unspecified location. The bounding box of a net is represented as a 4-tuple (*Xmin, Xmax, Ymin, Ymax*). For a horizontal cutline, they are *Ymin* and *Ymax*, and for the division of the cutline, they are *Xmin* and *Xmax*. As seen in Algorithm 4.2, a net's start and end points are used to index into *nets_start* and *nets_end* arrays when adding a net. The *nets_start* and nets_end methods make it easy to find the nets that come before and after a specific cutline. The list of nets that follow the n-th cutline is represented by the sum of *nets_start[i]* from *I = n* to the end. Similarly, from *I = 0*, *nets_end[i]* expands the list of nets before the n-th cutline.

The *total_workload_before* and *total_workload_after* arrays are one-dimensional. Each element at index i in these arrays represents the total workload of all nets that lie before the cutline at position i+1 and after the cutline at position i, respectively. There are two processes involved in creating these arrays. While working over the list of nets iteratively, the first step is completed. The burden of each net is added to the components of total_workload_before and total workload_after at indices that correspond to the start and conclusion of the net's bounding box, respectively, based on an estimate of the number of sinks. To get the final values, the second stop supplies in-place prefix sums to both arrays. The initial element for total_workload_before and the last element for total_workload_after are used to compute the prefix sums.

Determining the index opt_cut that minimizes the difference in absolute terms between total_workload_after [opt_cut +1] and total_workload_before [opt_cut]. is the initial stage after arrays are formed in figuring out the optimal cutline. Use *nets_before* = sum of *nets_end* [i] from I = 0 to opt_cut and *nets_after* = sum of *nets_strart*[I] from I = opt_cut + 1 to the end to extract the nets before and after the cutline depending on opt_cut.

Nets not in *nets_before* or *nets_after*, commonly referred to as *nets_crossing*, are included in the parent node of the two child nodes with nets in *nets_before* and *nets_after*. For nodes with *nets_before* and *nets_after*, the bi-partitioning procedure is repeated.

Algorithm 4.2: A Fast Heuristic for Optimal Cutline Selection

1. **procedure** get_optimal_cut(bbox, nets)
2. nets_start ← {{}}
3. nets_end ← {{}}
4. tot_work_ld_before ← {0}
5. tot_work_ld_after ← {0}
6. **for** net ϵ nets **do**
7. nets_start[net.start] ← nets_start[net.start] U net
8. nets_end[net.end] ← nets_end[net.end] U net
9. tot_work_ld_before [net.end] ← tot_work_ld_before[net.end] + net.work_ld
10. tot_work_ld_after [net.end] ← tot_work_ld_after[net.end] + net.work_ld
11. **end for**
12. tot_work_ld_before ← prefix sum of tot_work_ld_before starting from first element
13. tot_work_ld_after ← prefix sum of tot_work_ld_after starting from first element
14. optimal_cut ← -1
15. min-diff ← ∞
16. **for** cut ϵ cut indices **do**
17. diff ← [tot_work_ld_before[cut] – tot_work_ld_after[cut+1]]
18. **if** diff < min_diff ← **then**
19. min_diff ← diff
20. optimal_cut ← cut
21. **end if**
22. **end for**
23. **return** $\bigcup_{i=0}^{opt\_cut}$ nets_end[i] $\bigcup_{i=opt\_cut+1}^{last}$ nets_start[i]. opt_cut
24. **end procedure**

4.2 FEATURE SELECTION

Based on existing knowledge of placement and routing techniques, we choose a list of attributes that were available after placement and correlated with routing outcomes. Specifically, the following features were included:

1) Pin Density: The number of pins in a region of the FPGA.
2) Pin Accessibility: The potential number of wires connecting to the pins in each partition of the FPGA
3) Pin locations: The location of the pins is available after placement
4) Routing Demand Unit (RUDY): The total uniform wire count in bounding boxes of nets [12] calculated after placement
5) Bounding Boxes: The total number of the net bounding box outlines passing each partition
6) Net Density: The uniform wire volume in each routing channel
7) Wirelength: Total number of wires calculated after placement
8) Path Delay: Maximum delay of any path in the design
9) Worst Slack: The worst-case timing slack in the design
10) DRC Violation: The number of DRC (Design Rule Check) violations from all FPGA regions (partitions). It is the prediction target and the label for training
11) Congestion: Congestion is generated after placement or during initial routing and provides an estimation of the initial routing congestion map

4.3 EXPERIMENTAL DESIGN

4.3.1 Benchmarks

For the experiments, we used 13 benchmark designs collected from the VTR MCNC benchmark set (*Verilog to Routing*, n.d.). Table 4.1 presents the benchmark data for each benchmark design, including the total wirelength, the minimum channel width needed to route the design, and the number of 6-input Lookup Tables.

Table 4.1: Benchmarks for Evaluating Routability (Verilog-to-Routing, n.d.)

Benchmarks	#6-LUTs	Minimum Channel Width	Wirelength
apex2	1878	46	17963
alu4	1522	48	11268
apex4	1261	64	12735
des	1591	42	38803
bigkey	1699	34	23811
clma	8364	102	90179
diffeq	1494	52	13016
dsip	1362	32	23215
elliptic	3602	78	52166
ex1010	4598	78	39595
frisc	3539	82	47490
spla	3690	82	38469
tseng	1046	52	9875

The new VPR architecture modelling capabilities make it possible to model modern complex FPGA devices such as Altera Stratix family. In this study, we use a VPR architecture capture of Intel Altera Stratix IV. The device grid layout was altered Figure 4.3 to more accurately depict real Stratix IV devices. PLLs are placed at the edge of the I/O, and the I/O is configured so that they may reach any point for both vertical and horizontal routing. The DSP blocks, which contain multipliers and accumulators, as well as the M9K and M144K RAM blocks, are stored in columns. The remaining space on the device is used by Logic Array Blocks (LABs), which are composed of flip-flops (FFs), adders, and look-up tables (LUTs) (Verilog-to-Routing, n.d.).

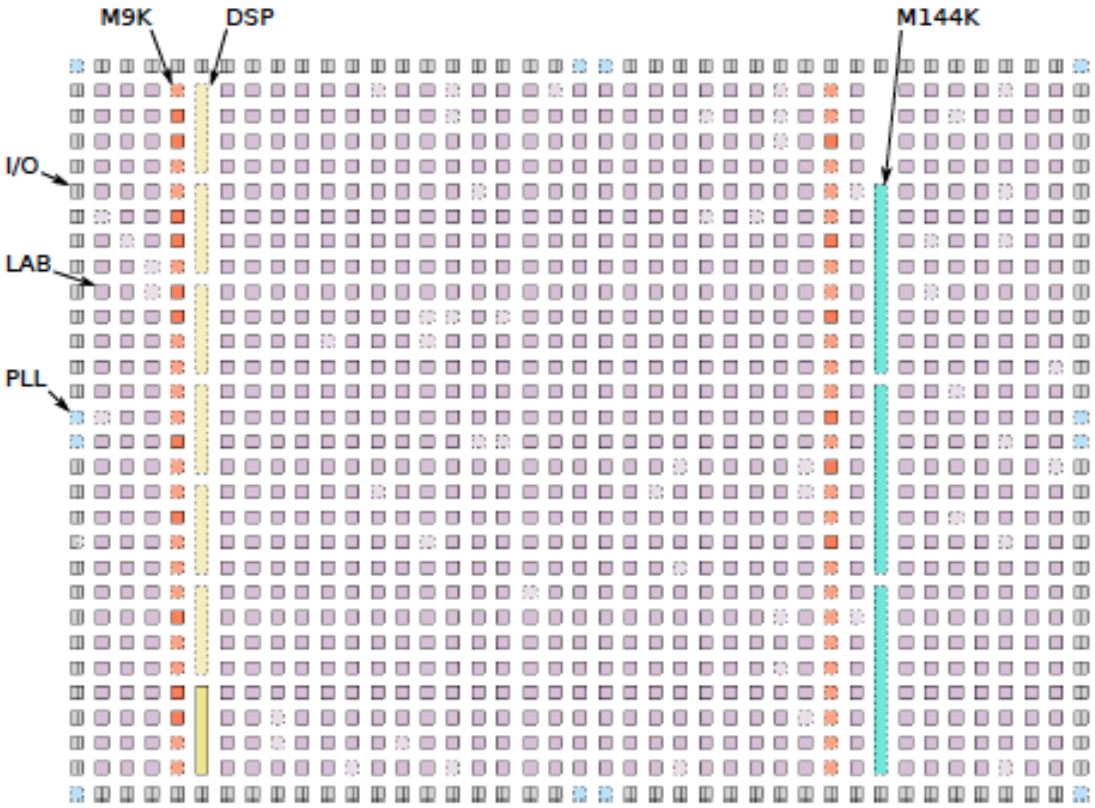

Figure 4.3: Example of Grid Layout and Block in VPR (Verilog-to-Routing, n.d.)

4.3.2 Routing Network

Based on a mix of 13% L16 (length 16) and 87% L4 (length 4) wires with a channel width of 300 tracks across the device, the Stratix IV routing network is displayed in Table 4.2. L16 wires can only be accessed through L4 wires; block pins cannot drive L16 wires or be driven by them. Furthermore, L16 cables only need switch blocks to join to other lines every fourth grid tile. The aspect ratio of the L4 driver input Muxes (DIMs) is 12:1. The input distribution of these muxes is as follows: six come from block pins (designated by Fc), while the remaining six come from a combination of additional L4 and L16 wires. Stratix IV uses larger muxes for the L16 DIMs to increase the total number of paths that can be added to the long-wire network. In the three-sided routing architecture of Straitix IV, a block can be connected to a routing channel that is situated above, below, or to the right of it. The direct-link connection paradigm includes links between different types of blocks. More

specifically, multi-height blocks like DSPs and M144K RAM blocks link to nearby blocks—which may have various heights—correctly, as shown in Figure 4.5 (*Verilog to Routing*, n.d.).

Table 4.2: Summary of Modelled Stratix IV Routing Architecture (Verilog-to-Routing, n.d.).

Metric	Expression	Value
Channel Width	W	300
L4 Wires	0.8667 - W	260
L16 Wires	0.1333 - W	40
L4 F_{cin}	$L4_{Fcin}$	0.055
L16 F_{cout}	$L4_{Fcout}$	0.075
L16 F_{cin}	$L16_{Fcin}$	0.000
L16 F_{cout}	$L16_{Fcout}$	0.000
L4 Driver Mux Size	$L4_{DIM}$	12:1
L16 Driver Mux Size	$L16_{DIM}$	40:1
L4 inter-switch-block distance	$L4_{ISBD}$	1
L16 inter-switch-block distance	$L4_{ISBD}$	4

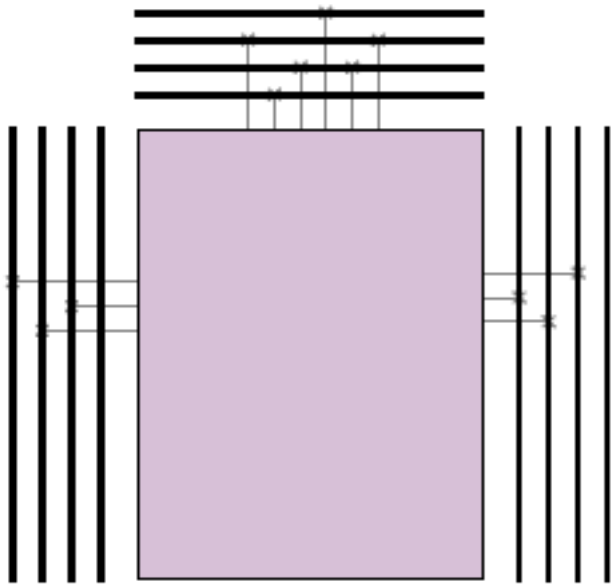

Figure 4.4: Three-sided Design with One Horizontal Track and One Vertical Track Accessible to Each Pin (Verilog-to-Routing, n.d.).

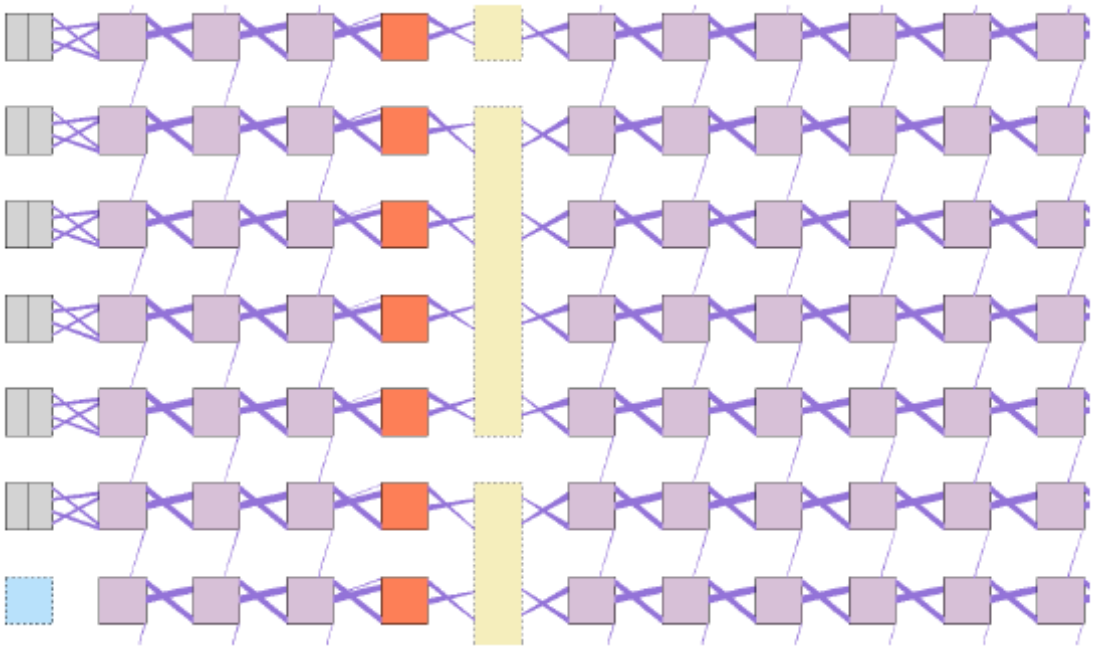

Figure 4.5: Direct Connections between Neighboring Blocks are indicated in Purple (Verilog-to-Routing, n.d.).

4.4 DATA COLLECTION AND ANALYSIS

Data Collection is one of the important steps in any research, as it forms the foundation for research findings. In this thesis, we use the benchmarks mentioned in Section 4.3.1, as shown in Table 4.1. With these benchmarks, we run different physical designs using VPR to generate different placement, routing and log files. These files provide the necessary data for our experiments. After collecting the data from these files using a Python script, we extract the relevant parameters. We then analyze this collected data using Python libraries like Pandas and Numpy to make them capable of being used in the experiments. The next chapter details the data structures used for each experiment. The process is summarized in Figure 4.6.

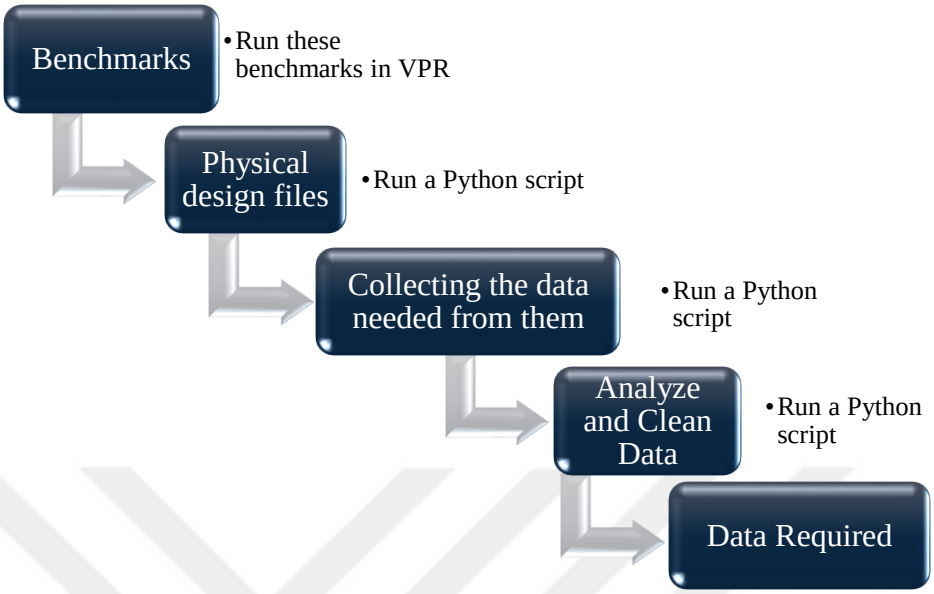

Figure 4.6: The Process for Getting Required Data

4.5 DRC VIOLATION PREDICTION

DRC violation prediction helps ensure that the design complies with the FPGA's architectural and manufacturing constraints. By identifying potential violations early in the design process, designers can make adjustments to maintain design integrity. In this work, we aim to predict DRC violations by using path delay, wirelength, and setup Worst Negative Slack during the placement phase as input features. We predict the first, fifth, and last iterations of conflict. Figure 4.7 shows the proposed model for our regression-based DRC violation prediction.

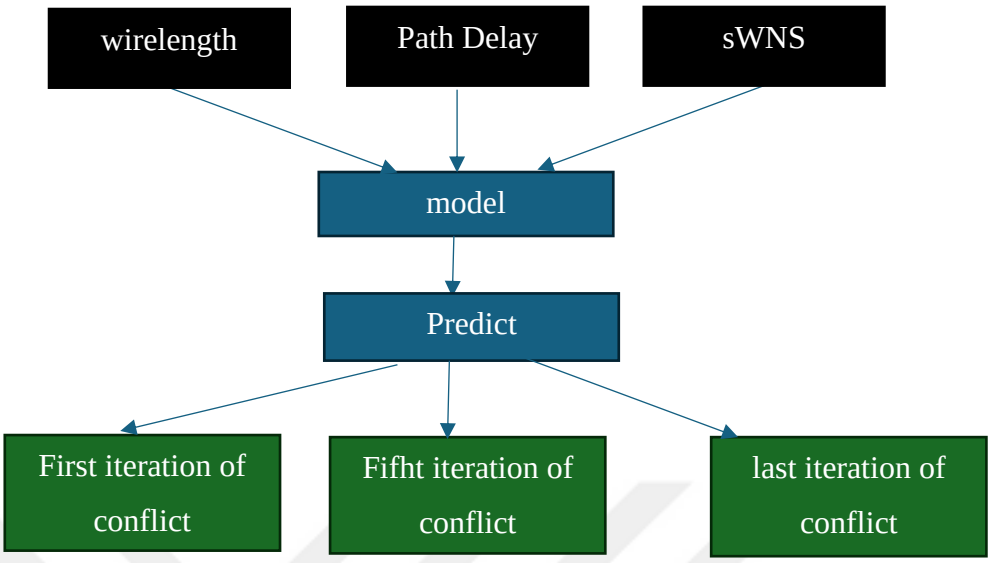

Figure 4.7: VRC Violation Prediction Model Flowchart

The training model in Figure 4.7 takes wirelength, path delay, sWNS as an input and first iteration of conflict, fifth iteration of conflict and last iteration of conflict as an output. The experiments and results will be discussed in the next chapter.

4.6 CONGESTION PREDICTION

Routing congestion prediction is a crucial for timely and effective design closure. We will leverage a pre-trained model from TensorFlow Hub model (*TensorFlow Hub*, n.d.) to achieve reliable congestion prediction. You can find pre-trained machine learning models for your specific problem on TensorFlow Hub.

Transfer learning is the technique of utilizing a pre-trained machine learning model. It can be costly and time-consuming to build a machine learning model from scratch and train it on a large amount of data. By leveraging the knowledge gained from a pre-trained model, we can adapt it to our specific task of classifying different routing states.

We used a pretrained model for image classification based on convolutional neural networks as we knew our challenge was image classification (classifying distinct routing state) (*TensorFlow Hub*, n.d.).

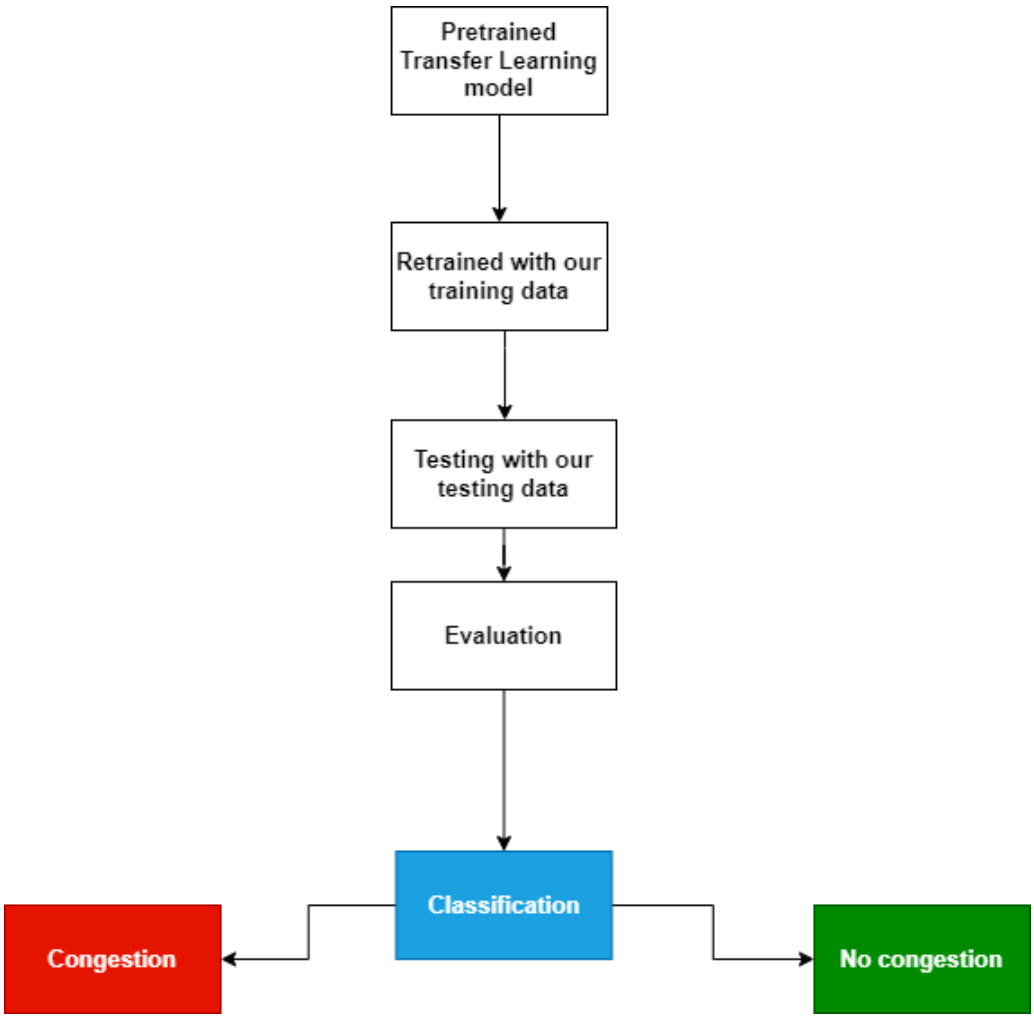

Figure 4.8: The Process of Congestion Prediction Model

Figure 4.8 illustrates the process of congestion prediction. The model takes an image as input, which is created from a routing file as described in Section 4.4. The model then classifies the image to predict whether it is congested or not.

4.7 ROUTIBILITY INFORMATION PREDICTION

Routibility prediction helps determine if a certain design can be successfully routed within the constraints of the FPGA architecture. Designers might avoid pursuing solutions that are unlikely to meet performance or area requirements by anticipating routibility in the early design process. In this work, we aim to predict three key variables in the routing phase:

- **Channel Width:** Channel width in physical design describes the number of tracks available in a routing channel between rows or columns of logic blocks.
- **F_{max}:** The term F_{max} in physical design refers to the highest operating frequency.
- **Setup Worst Negative Slack (SWNS):** Identifies the most critical violation of the setup timing constraint during the routing phase.

We predict these three variables using a machine learning regression model, using placement variables like wirelength, path delay and the violation of DRC by taking different iterations like the first, fifth and the last.

4.8 MODEL SELECTION

Model selection is an essential step in the overall effectiveness and success of the proposed AI solutions. The first step in model selection is problem definition. In this work, we deal with two different machine learning problems: classification problem for predicting congestion and regression problem for predicting DRC violation and routibility information.

4.8.1 Classification Model

Selection of the best model for machine learning classification task requires considering several factors, including the type of data, the problem complexity and the require performance. In this work, we are dealing with unstructured data (images) to classify whether an image is congested or not congested. Several deep learning models along with traditional machine learning models are suitable for this kind of problem, including:

- Support Vector Machines (SVMs)
- K-Nearest Neighbors (KNN)
- Convolutional Neural Network (CNN)
- Transfer Learning

We could use Convolutional Neural Network (CNN) are generally one of the best choices for image classification task due to its ability to recognize the patterns within images, such as congestion. However, CNNs require a large amount of data for good performance. Since we may not have a large dataset, a recommended approach is to use transfer learning with a pre-trained CNN model, such as ResNet or MobileNet, which have been trained on massive datasets like ImageNet. By fine-tuning these pre-trained models on our specific image dataset, we can achieve good performance without requiring a large amount of training data.

4.8.2 Regression Model

Selecting the appropriate regression model depends on various factors, including the characteristics of your data, the relationship between features and target variables, and the specific requirements of your task. In this work, we compare three regression models on our data to determine the best model for our regression problem. Table 4.3 presents the machine learning performance metrics for each model after training.

Table 4.3: Comparison between Regression Models

	MAE	MSE	RMSE	R^2
Linear Regression	1.72	5.78	2.40	0.98
Decision Trees	2.25	13.24	3.63	0.96
Random Forest	1.91	9.11	3.01	0.97

As shown in Table 4.3, Linear Regression outperforms other models across all machine learning regression metrics. Therefore, we will employ Linear Regression as the regression model for this work.

5 EXPERIMENTS AND RESULTS

This section details the experimental setup, including dataset construction, feature engineering, NAS approach, and training details. Next, we present our evaluations on two benchmark datasets for routability prediction:

- Congestion in route
- Hotspot detection using DRC
- Routing information prediction

The following section presents the different experiments conducted using various Machine Learning models to predict routability.

5.1 EXPERIMENT 1: ROUTING CONGESTION PREDICTION

In this experiment, we aim to predict routing congestion by converting routing files into images. These images are generated by visualizing the wire connections as lines on a 2D plane, using coordinate information extracted from the routing files. To collect our dataset, we employed VPR to generate 200 routing files for a specific benchmark within the vtr_flow framework, utilizing the seed option to introduce variability. Subsequently, we developed a Python script to convert these files into images and assigned labels to each image based on its congestion level, as illustrated in Figure 5.1.

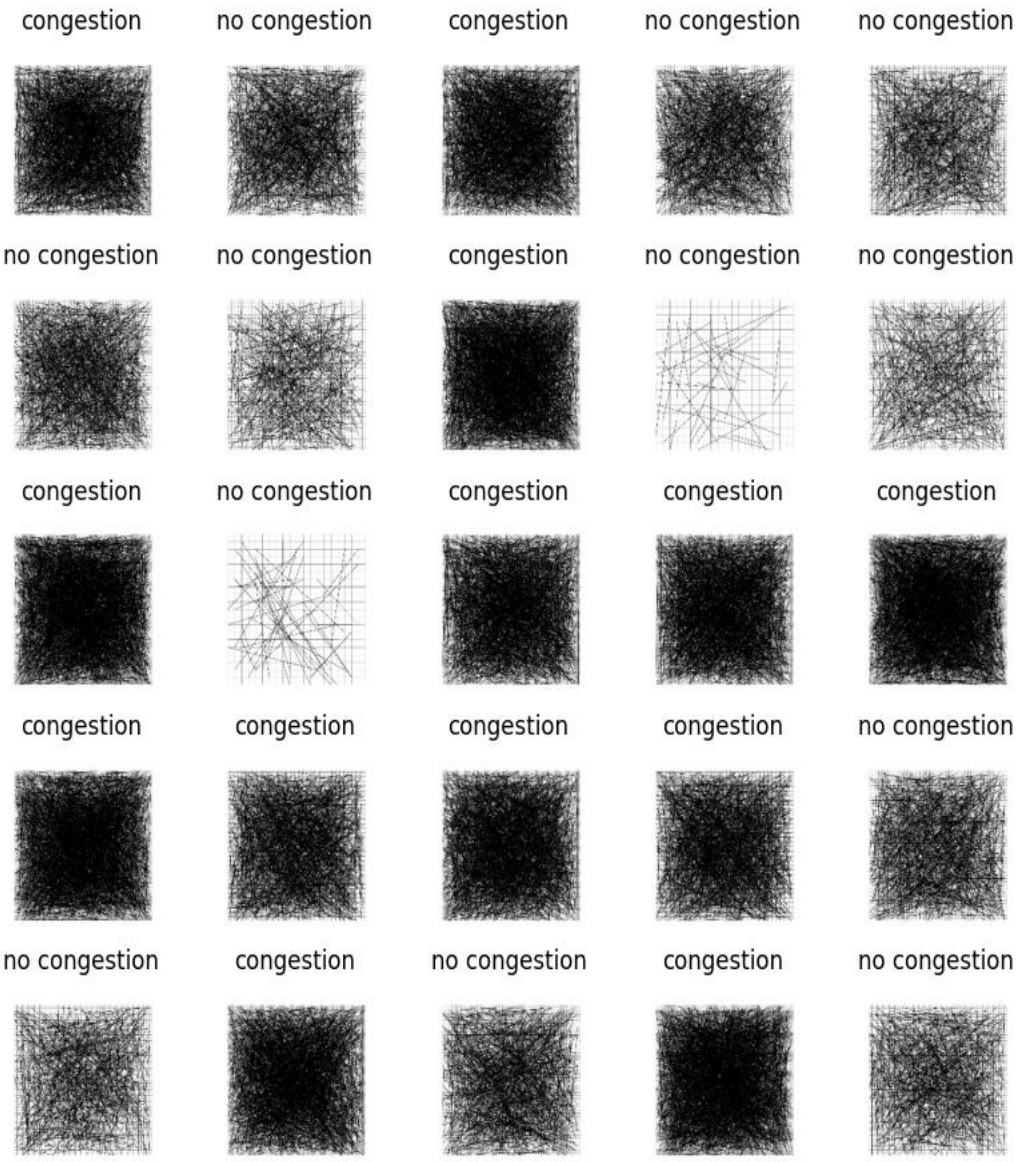

Figure 5.1: Format of Dataset for Congestion Prediction Model

In this experiment, we decided to use transfer learning technique. Transfer learning is a machine learning approach where a model trained on one task is repurposed as the starting point for a model on a related task. Consequently, we utilized a pre-trained classification model called MobileNetV3small from TensorFlow Hub (*TensorFlow Hub*, n.d.). MobileNetV3small is a lightweight, efficient Convolutional Neural Network (CNN) model made for embedded and mobile devices. By optimizing for both speed and accuracy,

it expands upon the MobileNet family and is perfect for real-time applications. We retrained the MobileNetV3small model with our data and tested its performance. Table 5.1 shows the classification report performance for our congestion prediction model where we have the model performances for classification models: Precision, Recall, F1-score and Accuracy.

Table 5.1: Performance Analysis of the Routing Congestion Prediction Model

	Precision	Recall	F1-score	support
Congestion	0.79	0.75	0.77	20
No congestion	0.76	0.80	0.78	20
Accuracy			0.78	40
Macro avg	0.78	0.78	0.77	40
Weighted avg	0.78	0.78	0.77	40

5.2 EXPERIMENT 2: PREDICTING DRC VIOLATION

DRC violation guarantees that the design satisfies manufacturing specifications and won't cause a chip failure. It is a crucial component of the physical design cycle. DRC breaches can be the result of several issues, including poor component placement, misrouted interconnects, or insufficient element spacing. In this experiment, we are trying to predict DRC violation by the variation of number of conflicts during routability, we selected a Linear Regression as the model and trained it with the data described in Table 5.2.

The model uses wirelength, path delay, and sWNS as inputs, and first iteration of conflict, fifth iteration of conflict, and last iteration of conflict as outputs. We choose these iterations because the violation always changes in these numbers of iterations. We used successfully routed designs to obtain the input parameters because the vpr_stdout.log file did not contain the necessary input parameters for unrouted designs.

Table 4.2: Dataset Format for Predicting DRC Violation Model

Wirelength	Path Delay	sWNS	First Iter conflict	Fifth Iter conflict	last Iter conflict
11462	5.70193	-5.70193	2548	1401	0
5141	9.39175	-9.39175	1192	625	0
6388	9.29765	-9.29765	1239	760	0
28067	10.6665	-10.6665	1645	806	0
17991	6.06831	-6.06831	1376	264	0
49310	9.24875	-9.24875	8188	3373	0
6583	6.93952	-6.93952	1192	459	0
17722	6.18984	-6.18984	1363	293	0
24178	8.57065	-8.57065	3194	1512	0
11462	5.70193	-5.70193	2548	1401	0

The model used 200 vpr_stdout.log files as a dataset, with 75% of the data used for training and 25% for testing. After training and testing, we evaluated the model's performance to see how good it is by comparing the predicted values and the actual values. Tables 5.3 and 5.4 show the machine learning performance metrics for the model.

Table 5.3: Predicted Value vs the Actual Value for DRC Violation Model

Predicted first Iter conflict value	Predicted fifth Iter conflict value	Predicted last Iter conflict value	Actual first Iter conflict value	Actual fifth Iter conflict value	Actual last Iter conflict value
8036	3350	0	8138	3416	0
8138	3266	0	8161	3200	0
8132	3316	0	8168	3307	0
8134	3400	0	8159	3350	0
8031	3334	0	8018	3236	0
8165	3364	0	8172	3466	0
8242	3423	0	8101	3503	0

8212	3416	0	8157	3350	0
8155	3333	0	8201	3420	0
8125	3328	0	8158	3479	0

Table 5.4: Model Performance Metrics for DRC Violation

Coefficient of Determination (R-Squared)	Mean Absolute Error (MAE)	Mean Squared Error (MSE)	Root Mean Squared Error (RMSE)
0.99	58.20	6481.80	74.83

Table 5.4 presents the metrics for the DRC violation model. While the R-squared value is nearly perfect, other metrics are not as strong as desired. Predicting three outputs requires a large dataset and significant computational resources to achieve optimal results.

5.3 ROUTING INFORMATION EXPERIMENTS

In this section, we predict routing information, including Setup Worst Negative Slack (SWNS), Channel Width, and F_{max}, as mentioned in the previous chapter.

5.3.1 The SWNS during routing experiment

In digital circuit design, SWNS identifies the most critical violation of the setup timing constraint during the routing phase. The discrepancy between a signal's required arrival time and its actual arrival time at a certain point in the circuit is known as slack in timing analysis. A negative slack value indicates that the signal arrives later than expected, potentially causing malfunctions at the intended clock speed.

In this experiment, we aim to predict the SWNS during the placement stage. We employ Linear Regression model trained with the data given in Table 5.5. The model takes wirelength, path delay, first iteration of conflict, fifth iteration of conflict, last iteration conflict as an input and SWNS as an output.

We developed a Python script that extracts necessary parameters from vpr_stdout.log files using the Pandas library. It is important to note that only successfully routed designs

were used to obtain the input parameters, as the log files for failed designs do not contain the required information.

Table 5.5: The Dataset Format for Predicting SWNS Model

Wirelength	Path Delay	sWNS	First Iter conflict	Fifth Iter conflict	last Iter conflict
11462	5.70193	-5.70193	2548	1401	0
5141	9.39175	-9.39175	1192	625	0
6388	9.29765	-9.29765	1239	760	0
28067	10.6665	-10.6665	1645	806	0
17991	6.06831	-6.06831	1376	264	0
49310	9.24875	-9.24875	8188	3373	0
6583	6.93952	-6.93952	1192	459	0
17722	6.18984	-6.18984	1363	293	0
24178	8.57065	-8.57065	3194	1512	0
11462	5.70193	-5.70193	2548	1401	0

The model used 200 vpr_stdout.log files as a dataset, with 75% of the data used for training and 25% for testing. After training and testing the model, we evaluated its performance by comparing the predicted sWNS values to the actual sWNS values. The results are shown in Table 5.6.

Table 5.6: Predicted Value vs Actual Value for SWNS Model

Predicted sWNS value	Actual sWNS value
-9.36582	-9.36788
-9.77382	-9.7612
-9.35184	-9.34765
-9.6975	-9.68673
-9.37652	-9.37636
-9.41638	-9.41526
-9.20852	-9.20798
-9.41233	-9.41291

-9.4311	-9.43575
-9.36487	-9.36498

Table 5.7: Machine Learning Performance for SWNS Model

Coefficient of Determination (R-Squared)	Mean Absolute Error (MAE)	Mean Squared Error (MSE)	Root Mean Squared Error (RMSE)
0.99	3.47	0.004	0.005

Table 5.7 presents the machine learning model performance metrics for the sWNS model, which indicate strong performance across all measures. These results suggest that our model is highly effective.

5.3.2 Minimum Channel Width Experiment

Channel width in physical design describes the number of tracks available in a routing channel between rows or columns of logic blocks, especially in the context of FPGA and ASIC design. The channel width determines its capacity to carry signal lines or interconnections. In this experiment, our aim is to predict the minimum channel width for a physical design, which helps to ensure that routing path was optimized for routing performance and improving overall the circuit speed. We will use a Linear Regression model, trained on the data shown in Table 5.8. It is important to note that the channel width must be an even number.

The model uses wirelength, path delay, sWNS, as an input and channel width as an output. A Python script was developed to extract these parameters from vpr_stdout.log files using the Pandas library. Only successfully routed designs were used to obtain the input parameters, as the vpr_stdout.log files for failed designs do not contain the necessary information. The results are presented in Table 5.9.

Table 5.8: Dataset Format for Predicting Minimum Channel Width Model

Wirelength	Path delay	sWNS	Channel width
11462	5.70193	-5.70193	46
5141	9.39175	-9.39175	48
6388	9.29765	-9.29765	64
28067	10.6665	-10.6665	42
17991	6.06831	-6.06831	34
49310	9.24875	-9.24875	102
6583	6.93952	-6.93952	52
17722	6.18984	-6.18984	32
24178	8.57065	-8.57065	78
22558	10.6439	-10.6439	78

Table 5.9: Predicted Value vs Actual Value for Predicting Minimum Channel Width Model

Predicted Channel width value	Actual channel width value
50	50
48	48
48	46
48	48
102	100
102	102
102	100
102	102
102	102
102	102

The model used 200 vpr_stdout.log files as a dataset and trained with 75% as a training data and 25% for testing and validation. After training and testing, we evaluated the model to see how good it is by comparing the predicted value of minimum channel width and the actual value of minimum channel width with the results shown in Table 5.10.

Table 5.10: Machine Learning Performance Measures for Minimum Channel Width Model

Coefficient of Determination (R-Squared)	Mean Absolute Error (MAE)	Mean Squared Error (MSE)	Root Mean Squared Error (RMSE)
0.99	0.59	0.38	0.61

As shown in Table 5.10, the high R-squared value indicates a strong correlation between predicted and actual values. However, the relatively low values of MAE, MSE, and RMSE suggest that the model might suffer from overfitting or data quality issues.

5.3.3 F_{max} experiment

In physical design, the term F_{max} refers to the highest operating frequency at which a digital circuit can reliably function. This metric is particularly important during the placement phase, as it indicates the maximum clock frequency that the design can accommodate without violating timing constraints.

In this experiment, we aim to predict F_{max} during placement stage which is critical to ensure that the design meets its performance requirements. We employ a Linear Regression model, trained on the dataset presented in Table 5.11. The model utilizes wirelength, path delay, and SWNS as input features and F_{max} as the target output. We developed a Python script to extract these parameters from vpr_stdout.log files and organized the data using the pandas library.

We utilized successfully routed designs to extract the necessary input parameters, as the vpr_stdout.log file for failed routed designs did not contain the required information.

Table 5.11: Dataset Format for Predicting F_{max} Model

Wirelength	Path delay	sWNS	F_{max} (MHZ)
11462	5.70193	-5.70193	136.934
5141	9.39175	-9.39175	89.552
6388	9.29765	-9.29765	99.27
28067	10.6665	-10.6665	88.6076
17991	6.06831	-6.06831	155.224
49310	9.24875	-9.24875	93.1046
6583	6.93952	-6.93952	131.047
17722	6.18984	-6.18984	156.07
24178	8.57065	-8.57065	97.5731
11462	5.70193	-5.70193	136.934

The model was trained on a dataset of 200 vpr_stdout.log files, with 75% of the data used for training and 25% for testing and validation. After training and testing, we evaluated the model's performance by comparing the predicted F_{max} values to the actual F_{max} values, as shown in Table 5.12.

Table 5.12: Predicted Value vs Actual Value for Predicting F_{max} Model

Predicted F_{max} value	Actual F_{max} value
94.63368	93.0224
94.60925	95.3351
99.10681	100.23
96.54104	96.87
96.71474	97.363
101.3002	100.6151
96.12709	96.84
99.98368	98.5985
96.34636	97.4251
96.83021	96.051

Table 5.13: Machine Learning Performance Measure for F_{max} Model

Coefficient of Determination (R-Squared)	Mean Absolute Error (MAE)	Mean Squared Error (MSE)	Root Mean Squared Error (RMSE)
0.84	3.28	16.86	4.10

Table 5.12 shows the machine learning performance measure for F_{max} model where we are predicting the F_{max} value. Based on the result in the table we could say the model has a great performance in the learning and in the prediction.

5.6 EVALUATION

To evaluate the performance improvements our approach brings to existing FPGA CAD tools, we tuned the routability predictions to a large-scale (450K LUTs) and highly congested design that previously failed to route due to congestion, as shown in Figure 5.1 within the rectangles. Notably, this design took about 14 hours before ultimately failing. Our tool was able to identify these congested areas, finding that the initial congestion area had a 75.00% usage rate, with 35.44% of the logic blocks placed in these bounding boxes being over-utilized, causing congestion.

Using these predictions, we updated the router's cost function to account for this congestion, specifically avoiding routing high fanout nets (signals with 500+ connections) in these partitions. As a result, the design took only 3 hour and 48 minutes to fully route (

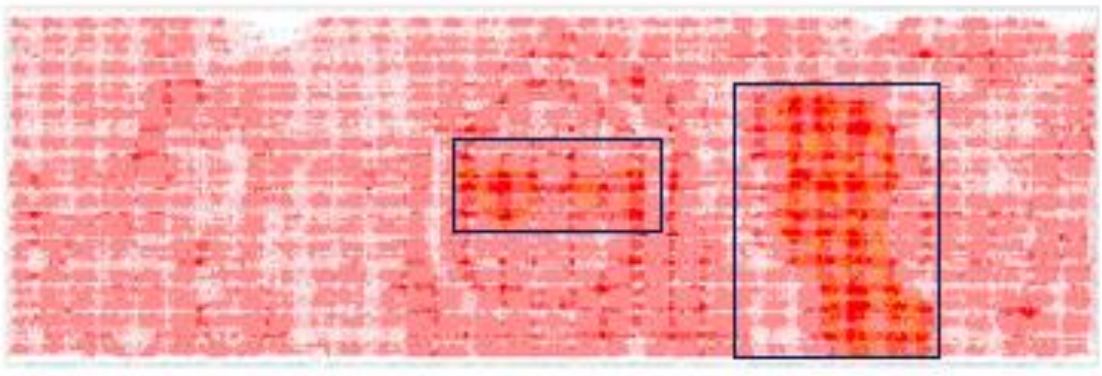

Figure 5.2: Visualization of a Highly Congested Design Captured Using Lattice Semiconductor Radiant FPGA Designer Tool (Lattice Radiant Software. (n.d))

6 CONCLUSION AND FUTURE WORK

The application of machine learning (ML) techniques in accelerating FPGA physical design tasks shows great promise, allowing FPGA CAD tools to learn from past experiences and solve current problems more efficiently.

In this study, we introduce a new framework for predicting routing congestion information in large-scale FPGA designs during the placement phase. Our approach employs a load-balanced bi-partitioning strategy to organize the input net list, enhancing the accuracy of routing information prediction. The framework utilizes carefully crafted features representing the placement scheme and design connectivity. This partitioning or clustering approach to the input net list advances the state of the art and significantly contributes to the model's success in accurately predicting most routing information. Furthermore, this load-balanced clustering approach improves routability for highly congested designs and significantly reduces router runtime.

Our proposed method achieved accuracy comparable to initial routing for total routability predictions, but with significantly shorter runtime. This effectively addresses the challenge of achieving both speed and accuracy in general routability prediction.

We utilize well-engineered features that describe the placement scheme and design connectivity, encoded in the input image, as input for our routing congestion prediction framework for FPGA designs at the placement stage. Compared to other approaches, our proposed strategy performs well. This is mostly the result of the partitioning strategy that was applied to group the input net list according to bounding box and connection.

Investigating FPGA routing designs at an early stage necessitates intelligence and rapid design iterations, which drives the investigation of quick, precise measurements. With great accuracy, our proposed model could forecast the worst slack delay, minimum channel width (W_{min}), and maximum attainable frequency (F_{max}) for FPGA designs. We demonstrated that the routability estimations from our model correlate well with the output from the whole VPR CAD flow. Significantly, our model is much faster than traditional

VPR, and further speedups can be achieved with access to a GPU server, making it suitable as a tool for evaluating FPGA routability.

The future work should involve extending the power estimating model to physical synthesis and technology mapping, beyond its current use for power estimation. Wotan could be used to determine the minimal routable channel width of an architecture. Given a fixed demand multiplier, channel width might be adjusted as an alternative to modifying the demand multiplier to change congestion to a constant channel width.

Another potential area of future research is creating a new power model using the same methodology. Our model can compute relative routability metrics and tune performance with significantly less computational effort than the power model integrated into VPR, which optimizes power-delay without taking interconnect routability into account and requires an extremely high computational time when exploring a limited number of interconnect architecture axes.

REFERENCES

(1) Al-Hyari, A. (2019). *Towards Smart FPGA Placement Using Machine Learning*. https://atrium.lib.uoguelph.ca/xmlui/handle/10214/17377

(2) Bhardwaj, V. (2021). FPGA Design Flow and CAD. *SSRN Electronic Journal*. https://doi.org/10.2139/ssrn.3788117

(3) Chai, Z., Zhao, Y., Lin, Y., Liu, W., Wang, R., & Huang, R. (2022). CircuitNet: an open-source dataset for machine learning applications in electronic design automation (EDA). In *Science China Information Sciences* (Vol. 65, Issue 12). Science Press (China). https://doi.org/10.1007/s11432-022-3571-8

(4) Chan, W. T. J., Ho, P. H., Kahng, A. B., & Saxena, P. (2017). Routability optimization for industrial designs at sub-14nm process nodes using machine learning. *Proceedings of the International Symposium on Physical Design*, *Part F127197*, 15–21. https://doi.org/10.1145/3036669.3036681

(5) Chang, C. C., Pan, J., Xie, Z., Zhang, T., Hu, J., & Chen, Y. (2024). Toward Fully Automated Machine Learning for Routability Estimator Development. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, *43*(3), 970–982. https://doi.org/10.1109/TCAD.2023.3330818

(6) Chen, D., Cong, J., & Pan, P. (2006). FPGA design automation: A survey. In *Foundations and Trends in Electronic Design Automation* (Vol. 1, Issue 3, pp. 195–330). https://doi.org/10.1561/1000000003

(7) Chen, J., Kuang, J., Zhao, G., Huang, D. J. H., & Young, E. F. Y. (2020). PROS: A Plug-in for Routability Optimization applied in the State-of-the-art commercial EDA tool using deep learning. *IEEE/ACM International Conference on Computer-Aided Design, Digest of Technical Papers, ICCAD*, *2020-Novem*. https://doi.org/10.1145/3400302.3415662

(8) Coulson, N. C., Sotiriadis, S., & Bessis, N. (2020). Adaptive Microservice Scaling for Elastic Applications. *IEEE Internet of Things Journal*, *7*(5), 4195–4202. https://doi.org/10.1109/JIOT.2020.2964405

(9) Dai, Z. (2003). *Routability Prediction for Field Programmable Gate Arrays with a Routing Hierarchy*.

(10) Farooq, U., Marrakchi, Z., & Mehrez, H. (2012). FPGA Architectures: An Overview. In *Tree-based Heterogeneous FPGA Architectures* (pp. 7–48). Springer New York. https://doi.org/10.1007/978-1-4614-3594-5_2

(11) Goswami, P., & Bhatia, D. (2021). Congestion prediction in fpga using regression based learning methods. *Electronics (Switzerland)*, *10*(16). https://doi.org/10.3390/electronics10161995

(12) Grewal, G., Areibi, S., Westrik, M., Abuowaimer, Z., & Zhao, B. (2017). Automatic flow selection and quality-of-result estimation for FPGA placement. *Proceedings - 2017 IEEE 31st International Parallel and Distributed Processing Symposium workshops, IPDPSW 2017, 115-123.*

(13) Elfadel, I. M., Boning, D. S., & Li, X. (2019). *Machine Learning in VLSI Computer-Aided Design*. Springer.

(14) Institute of Electrical and Electronics Engineers, IEEE Council on Electronic Design Automation, Association for Computing Machinery, & ACM Special Interest Group on Design Automation. (n.d.). *2017 IEEE/ACM International Conference on Computer-Aided Design (ICCAD) : digest of technical papers : November 13-16, 2017, Irvine Marriot, Irvine, CA*.

(15) Kannan, P., Balachandran, S., & Bhatia, D. (n.d.). *fGREP-Fast Generic Routing Demand Estimation for Placed FPGA Circuits*.

(16) Kapre, N., Ng, H., Teo, K., & Naude, J. (2015). InTime: A machine learning approach for efficient selection of FPGA CAD tool parameters. *FPGA 2015 - 2015 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, 23–26. https://doi.org/10.1145/2684746.2689081

(17) Liu, S., Sun, Q., Liao, P., Lin, Y., & Yu, B. (n.d.). *Global Placement with Deep Learning-Enabled Explicit Routability Optimization*.

(18) Lou, J., Krishnamoorthy, S., & Sheng, H. S. (2001). *Estimating Routing Congestion using Probabilistic Analysis*.

(19) Mametjanov, A., Balaprakash, P., Choudary, C., Hovland, P. D., Wild, S. M., & Sabin, G. (n.d.). *Autotuning FPGA design parameters for performance and power*.

(20) Petelin, O. (2016). *CAD Tools and Architectures for Improved FPGA*

Interconnect.

(21) *Proceedings, Design, Automation & Test in Europe : Dresden, Germany, March 24-28, 2014.* (n.d.).

(22) Pui, C.-W., Chen, G., Ma, Y., Y Young, E. F., & Yu, B. (n.d.). *Clock-Aware UltraScale FPGA Placement with Machine Learning Routability Prediction*.

(23) Wang, L. C., & Abadir, M. S. (2014). Data mining in EDA -basic principles, promises, and constraints. *Proceedings - Design Automation Conference*. https://doi.org/10.1145/2593069.2596675

(24) Xu, X., Matsunawa, T., Nojima, S., Kodama, C., Kotani, T., & Pan, D. Z. (2016). A machine learning based framework for Sub-Resolution Assist Feature generation. *Proceedings of the International Symposium on Physical Design*, *03-06-April-2016*, 161–168. https://doi.org/10.1145/2872334.2872357

(25) Yang, S., Gayasen, A., Mulpuri, C., Reddy, S., & Aggarwal, R. (2016). Routability-driven FPGA placement contest. *Proceedings of the International Symposium on Physical Design*, *03-06-April-2016*, 139–143. https://doi.org/10.1145/2872334.2886419

(26) Zdun, U., Queval, P., Simhandl, G., Scandariato, R., Chakravarty, S., Jelic, M., Jovanovic, A., Queval, -j, Simhandl, G., Jelic, M., Jovanovic, A., & Queval, P.-J. (2023). Microservice Security Metrics for Secure Communication, Identity Management, and Observability. *ACM Transactions on Software Engineering and Methodology*, *0*(1), 16. https://doi.org/10.1145/3532183

(27) Verilog-to-Routing. (n.d.). Verilog-to-Routing: Open-source CAD tools for FPGA research. https://verilogtorouting.org

(28) Google Developers. (n.d.). TensorFlow Hub: Open-source Machine learning tools https://www.tensorflow.org/tutorials/images/transfer_learning_with_hub

(29) Lattice Radiant Software. (n.d.). Lattice Radiant Software: Tool for develops FPGA designs https://www.latticesemi.com/latticeradiant

(30) Zhou, Y., Vercruyce, D., & Stroobandt, D. (2020). Accelerating FPGA routing through algorithmic enhancements and connection-aware parallelization.

ACM Transactions on Reconfigurable Technology and Systems, *13*(4), 1–26. https://doi.org/10.1145/3406959

BIOGRAPHY

I am Cheikhsaadbouh Etfaghaoubeid a graduate student from Beykoz University master’s degree program in computer engineering. I received my bachelor's degree in computer science from University of Nouakchott in Nouakchott, Mauritania. I am interested in technology in general and specifically Artificial Intelligence and physical design so for that I chose to be my thesis in these field. This thesis is about enhancing physical design using AI. We are trying to predict routing congestion in placement phase using machine learning and deep learning algorithms and predict also the DRC violation.

APPENDIX

This appendix contains the repositories links of thesis experiments uploaded on Google Drive, we have two repositories one for congestion prediction experiment and the other one for DRC experiment and routing information prediction experiments.

DRC and Routing information experiments link:

https://drive.google.com/drive/folders/1PwqR415k_pdmXdxL7LqgY5Md9n-u6WdY?usp=drive_link

Congestion Prediction experiment link:

https://drive.google.com/drive/folders/1hSuH3SEMZqhzqzx3mD-FFyDjx3Z72bNH?usp=drive_link