A Machine Learning Model to Predict Maximum Clock Frequency During FPGA Placement

A Thesis

Presented to

The Faculty of Graduate Studies

of

The University of Guelph

by

Mohsen Fathi

In partial fulfilment of requirements

for the degree of

Master of Science

December, 2021

ABSTRACT

A Machine Learning Model to Predict Maximum Clock Frequency During FPGA Placement

Mohsen Fathi
University of Guelph, 2021

Advisor:
S. Areibi and G. Grewal

Field programmable gate arrays (FPGAs) are integrated circuits that allow the designer to configure them multiple times based on the application while having fast computing power. Placement is one of the most challenging tasks in the computer-aided design (CAD) flow of the FPGA. Timing-driven is a placement flow that aims to place a design in a way that reaches a specific clock frequency after routing. The actual frequency of a design is heavily dependant on the placement strategy. However, it can only be exactly determined after routing. Since current methods to estimate the post routing frequency during placement have their own limitations and might mislead the placer during the process, a novel machine learning framework was proposed in this thesis that predicts the post routing frequency of a placed design at each stage during the placement flow with the mean absolute percentage error of 1.72%. The proposed framework was integrated into the timing-driven flow of GPlace to dynamically predict frequency during the placement process and guide the placement strategy. Results showed that the maximum frequency that could be reached was increased by an average of 3% on all of the benchmarks.

Acknowledgements

I would like to express my sincere gratitude and appreciation to my advisors Dr. Shawki Areibi and Dr. Gary Grewal. Their guidance and motivation throughout my study was of great help to my research and writing this thesis. I would also like to thank my colleagues in the ISLAB: Pooria Esmaeili, Timothy Martin, and Roni Mittal. Your collaboration, support and encouragement are what made this work possible. Finally, a special thanks to my family. You have done so much to encourage and support me in both my studies and life, I cannot thank you enough.

To

my family

whose love and encouragement helped accomplish this

thesis.

Contents

1 Introduction **1**

1.1 Motivation 6

1.2 Contributions 6

1.3 Thesis Organization 7

2 Background **8**

2.1 FPGA CAD Flow 8

2.2 Analytical Placement 11

2.2.1 GPlace3.0 11

2.2.2 GPlace4.0 13

2.3 UltraScale FPGA Architecture 15

2.4 Benchmarks 16

2.5 Maximum Frequency of Operation 17

2.6 Machine Learning 19

2.6.1 Predictive Models 21

2.6.2 Feature selection 23

2.6.3 Hyperparameter tuning 24

2.6.4 Performance Metrics 24
2.7 Summary 26

3 Literature Review **27**
3.1 Machine Learning in CAD Flow 27
3.2 Frequency Prediction in ASICs 29
3.3 Frequency Prediction in FPGAs 30
3.4 Summary 32

4 Methodology and Proposed Framework **33**
4.1 ML framework 34
4.1.1 Benchmarks 34
4.1.2 Placer 36
4.1.3 Feature Engineering 36
4.1.4 Label Extraction 42
4.1.5 Training 44
4.1.6 Testing 47
4.2 Deployment 48
4.3 Case Studies 48
4.4 Summary 51

5 Results **52**
5.1 Experimental Setup 52
5.2 ML Framework Results 53
5.2.1 Model 1 53

5.2.2 Model 2 62
5.2.3 Model 3 71
5.2.4 Model 4 80
5.2.5 Analysis 89
5.3 Case Studies 91

6 Conclusions and Future Work **94**
6.1 Conclusion 94
6.2 Future Work 95

A Glossary **96**

Bibliography **98**

List of Tables

2.1 ISPD 2016 Placement Contest Benchmark Statistics 17

2.2 Range of Key Circuit Features 17

4.1 List of Resource Features 38

4.2 List of Connectivity Features 38

4.3 List of Longest Path Features 39

4.4 List of Congestion Features 40

4.5 List of Net-based Features 41

4.6 List of FPGA Utilization Features 41

4.7 List of Timing Features 42

4.8 Range of Hyperparameters for Each Model 46

5.1 Model 1 Feature Selection Result 54

5.2 Best Hyperparameters for Model 1 55

5.3 Model 1 Initial Results 56

5.4 Model 1 Final Results 57

5.5 Model 1 MSE for Each Group 58

5.6 Model 1 MAPE for Each Group 59

5.7 Model 1 Training and Inference Time 59
5.8 Model 2 Feature Selection Results 63
5.9 Best Hyperparameters for Model 2 64
5.10 Model 2 Initial Results 65
5.11 Model 2 Final Results 66
5.12 Model 2 MSE for Each Group 67
5.13 Model 2 MAPE for Each Group 68
5.14 Model 2 Training and Inference Time 68
5.15 Model 3 Feature Selection Results 72
5.16 Best Hyperparameters for Model 3 73
5.17 Model 3 Initial Results 74
5.18 Model 3 Final Results 75
5.19 Model 3 MAPE for Each Group 76
5.20 Model 3 MSE for Each Group 77
5.21 Model 3 Training and Inference Time 77
5.22 Model 4 Feature Selection Results 81
5.23 Best Hyperparameters for Model 4 82
5.24 Model 4 Initial Results 83
5.25 Model 4 Final Results 84
5.26 Model 4 MSE for Each Group 85
5.27 Model 4 MAPE for Each Group 86
5.28 Model 4 Training and Inference Time 86
5.29 Improvement after Performing Feature Selection (FS) and Hyperparameter Tuning (HT) 89

5.30 GPlace4.0 and GPlace4.1 Maxmimum Frequency Comparison on 12 Benchmarks 93

List of Figures

2.1 FPGA CAD Flow 9
2.2 GPlace3.0 flow 12
2.3 Xilinx UltraScale Architecture. 16

4.1 Proposed Framework 34
4.2 K-means on 372 Benchmarks 35
4.3 PCA on 372 Benchmarks 36
4.4 Frequency for all Placements 43
4.5 Dataset Correlation Heatmap 45
4.6 Comparison Between Post-routing and STA Estimation of Maximum Frequency 49
4.7 Integrating the ML model into GPlace Flow 50

5.1 Actual and Predicted Frequency for Model 1 60
5.2 Actual and Predicted Frequency for Model 1 61
5.3 Loss Function for Model 1 62
5.4 Actual and Predicted Frequency for Model 2 69
5.5 Actual and Predicted Frequency for Model 2 70

5.6 Loss Function for Model 2 71
5.7 Actual and Predicted Frequency for Model 3 78
5.8 Actual and Predicted Frequency for Model 3 79
5.9 Loss Function for Model 3 80
5.10 Actual and Predicted Frequency for Model 4 87
5.11 Actual and Predicted Frequency for Model 4 88
5.12 Loss Function for Model 4 88
5.13 Comparison Between Post-routing and STA Estimation of Maximum Frequency 92

Chapter 1

Introduction

In the past few decades, *General-Purpose Processors* (GPP) and *Application-Specific Integrated Circuits* (ASIC) were the main frameworks used to satisfy computational needs. ASICs are capable of providing efficient, high performance computations, but they take a long time to get to market, they have an expensive design cycle, and they are not reconfigurable. In comparison, GPPs are very flexible since they can perform a wide range of applications. However, they suffer from a low performance and clock rate. With the rise of new applications, including large-scale optimization problems [1], machine learning (ML) algorithms [2], and real-time systems [3], the gap between ASICs and GPPs has become critical, leading to the need for reconfigurable systems. A *Field Programmable Gate Array* (FPGA) is a type of integrated circuit that has been introduced to bridge this gap. A circuit designer can reconfigure an FPGA for different applications multiple times after manufacturing. FPGAs permit more flexibility and less time to market than ASICs while providing a faster clock frequency than GPPs. These benefits have made FPGAs a suitable frameworks for a wide range of applications, ranging from video and

image processing to aerospace and military equipment. However, FPGAs have their own challenges.

To implement a design on an FPGA, *Computer-Aided Design* (CAD) tools are used to reconfigure the FPGA to a suitable hardware specification for an application. During the CAD flow, the design goes through different stages to map a circuit from a human-readable form to a form that can be downloaded on the FPGA to reprogram it. This process starts with logic synthesis, in which a netlist of the circuit is created. This netlist consists of nodes and nets. Each node represents a component on the FPGA, while the net shows the connectivity of the different components. As well, during this step various optimization techniques are used to simplify the logic, and reduce the number of gates and the complexity of the circuit. Placement is the next step that is performed and is responsible for assigning each cell in the netlist to a legal location on the FPGA while reducing an objective function. This function could be the total estimated wire length, the maximum frequency of operation, or the routability. Finally, during routing, the router decides how to connect different components of the circuit by configuring wire segments in the FPGA.

Due to wide commercial adoption, FPGA architectures have evolved significantly over the last two decades. Modern FPGA architectures are comprised of heterogeneous blocks, like LUTs, FFs, BRAMs and DSPs. Each of these components have strict constraints on their usage. In addition to their complexity, modern FPGAs are very large, as are the applications mapped onto them. These two factors have made the placement and route stages very challenging.

Short design cycles are one of the main goals in FPGA design. During CAD flow, the circuit goes through different steps, but placement and routing are the most time-

consuming steps. As the size and complexity of the FPGAs started to increase over time, the expected place-and-route time became more critical; for today's large designs, it may take hours or even days. Placement and routing, which are naturally related to each other, are treated as separate problems, due to their NP-hard complexity [4]. This results in a gap between these two stages. In addition to a high runtime, the quality of the placement is also very important since it can affect many parameters including the maximum frequency of operation (F_{max}) or the routability; however, these parameters cannot be precisely determined until the end of the routing stage.

Timing-driven placement is a flow that aims to achieve a specific clock frequency. During the flow, the criticality of the paths is first determined. The placer then tries to reduce the interconnect delay by minimizing the most critical timing paths. The final clock frequency of a design, however, cannot be known until the routing step decides which routing resources on the FPGA should be used to connect the cells. Routing solutions that fail to achieve timing closure may require many iterations of place-and-route at a very high computational cost. This provides an impetus for determining the post-routing maximum clock frequency of a design during placement.

During timing-driven placement, *Static Timing Analysis* (STA) is performed to identify timing-critical paths. This loosely couples the placement and routing steps; however, STA has its own drawbacks. First, as the routing resources on the FPGA are fixed, large designs that require large numbers of resources must be spread out over the FPGA fabric to access the required resources. This has the effect of reducing the accuracy of the delay model, as the delay becomes dominated by interconnect (harder to assess) rather than logic (easier to assess). Second, the routing delays used in STA do not typically take congestion into account. Congestion can have a significant effect on path delay, as

the router must search for alternative paths in regions of high congestion. This can result in the placer and the router identifying different critical paths. Third, STA does not typically take the behavior of the router into account. This can also lead to the placer optimizing different critical paths than the router. Fourth, modern FPGAs are heterogeneous and contain regions of special logic blocks, like DSPs and BRAMs. These regions often give rise to congestion due to high demands and the limited availability of routing resources. The resulting inaccuracies in the estimation of timing delays can mislead the placers when the gap is very large. This can result in a placement solution that, when routed, fails to meet the target clock frequency for the design.

As the routing and placement stages are dependent on each other, it is important to know whether the current placement is able to satisfy all the timing requirements after routing. The ability to predict the actual critical path delay of a design during placement can help the placer to optimize the timing delays of the design more efficiently. It will also be more likely to achieve its timing closures after routing. As feedback is only available to the placer after the time-consuming step of routing is performed, this thesis proposes a model that will accurately and efficiently estimate the maximum clock frequency during placement. An accurate estimation of the frequency in the early stages of the placement can help the placer to improve its optimization strategy. For example, if the desired timing requirements are reached during the placement, more optimizations can be avoided, thus reducing the runtime. In comparison, if the desired frequency cannot be reached during placement, the placer can be guided to increase its optimization effort. If, in the end, a solution is obtained that cannot reach the target frequency, the time-consuming stage of routing can be avoided, and a different placement flow can be used to achieve the timing goal.

Artificial Intelligence (AI) refers to computers and machines that mimic the problem-solving and decision-making capabilities of a human. The goal of AI is to design systems that think and act like humans. Machine learning (ML) is a subset of AI that is based on the idea of learning from the data. ML finds patterns and structures in the data to predict an unknown variable with minimal human interaction. The high degree of accuracy in ML, combined with its ability to be trained offline and its efficiency in comparison to traditional problem-solving approaches, have made it a good candidate for many applications.

Today, with the drastic rise in computational power and the availability of data and cloud systems, machine learning is integrated into all daily applications. ML is widely used in various areas like image processing, the military, and medicine. The need for fast and efficient design time for FPGAs have made CAD tools a candidate for machine learning models. Recent works like [5] have integrated ML in the CAD tools to predict the routability of the design during placement and were able to improve the runtime and reduce unroutable solutions. Other works like [6] have addressed congestion, which is another challenge related to the CAD flow. These works have proposed an ML framework to predict congestion during the placement. Due to the importance of F_{Max} and the gap that exists for an accurate model, this area could be another candidate for the application of machine learning.

This thesis has developed several machine learning models to predict the maximum frequency of operation during placement. These models were integrated into the placement flow to provide a fast and accurate feedback for the placement engine. The thesis ends with a discussion of the effectiveness of several case studies using F_{Max} in the placement engine.

1.1 Motivation

The use of post routing timing information during the FPGA placement involves several challenges and is motivated by several factors:

1. Although routing and placement are treated as separate problems, they can affect each other. The quality of the placement can determine many important parameters of the design after routing, such as frequency. Although probes are needed to capture this dependency during placement, an accurate prediction of these parameters is a still a challenging task.

2. In many cases STA, which is widely used as the delay model in timing driven placement, is unable to produce accurate estimations for the F_{max} of the placed design. STA does not consider several important factors, such as congestion and the behavior of the router. The router could thus fail to achieve its timing closure.

3. Placement and routing time increase as the size and complexity of the FPGA rise. Optimizing the placement by predicting the post routing probes, such as routability, F_{max}, and congestion, can help to reduce total place and route time.

1.2 Contributions

The main contributions of this thesis are as follows:

1. Engineering fast and efficient features to use in an ML model to predict the maximum frequency of operation during placement. These features represent the different static and dynamic characteristics of the benchmarks and the current placement.

2. Applying a set of simple ML models, such as linear regression [7], support vector machines (SVM) [8] and a decision tree [9], alongside ensemble models, such as XGBoost [10], random forest [11], and stacking models [12], to predict the maximum frequency of operation during placement with a high degree of accuracy.

3. Integrating the frequency prediction framework into GPlace4.0 to produce an accurate frequency for the current placement and to guide the placer engine to improve the quality of the results.

1.3 Thesis Organization

Chapter 2 presents the necessary background information. This includes the CAD flow and placement stage for the FPGAs, the different challenges and models used in machine learning, the target FPGA architecture, the placement tool used in the simulations, the information about the maximum frequency of operation for FPGAs, and the benchmarks that were used to train the model. Chapter 3 contains a literature review of the previous work done in this area. It focuses on integrating ML into CAD, and on frequency prediction in ASICs and FPGAs. Chapter 4 involves a discussion of the proposed framework and methodology, including the ML frequency prediction framework and its integration into the GPlace4.0 flow. Chapter 5 includes the results and simulations. Finally, Chapter 6 provides a summary of both the work done in this thesis and the results.

Chapter 2

Background

This chapter provides the background required to understand the topics introduced in this thesis. Section 2.1 presents an overview of the FPGA CAD flow, and details regarding the different steps. Section 2.2 presents the analytical placement and the GPlace placement tool, as well as its wirelength and timing driven flows. Section 2.3 presents the FPGA device architecture that is targeted in this thesis. Section 2.4 presents the benchmarks that were used for data preparation and creating the dataset. Section 2.5 presents information about the calculation of F_{max} in digital design and timing constraints. Section 2.6 introduces both background information and concepts in machine learning, as well as the details of the models and performance metrics used in this thesis.

2.1 FPGA CAD Flow

FPGAs are ICs that have the capability to reconfigure their hardware many times for different applications. FPGAs have different logic blocks that can be programmed to per-

form different functions. They also contain programmable wire segments and switches to connect different components on the FPGA to each other. The first step in programming an FPGA is to define the circuit using a method of design entry. Hardware description language (HDL) is a category of programming languages that can be used to define the functionality of the circuit. However, the HDL design cannot be directly fed to the FPGA in order to program it. The design should go through a CAD flow to produce a bitstream that can be used to configure the FPGA. Figure 2.1 shows an overview of the FPGA CAD flow. Although the design goes through several steps in the CAD flow, there are three main steps:

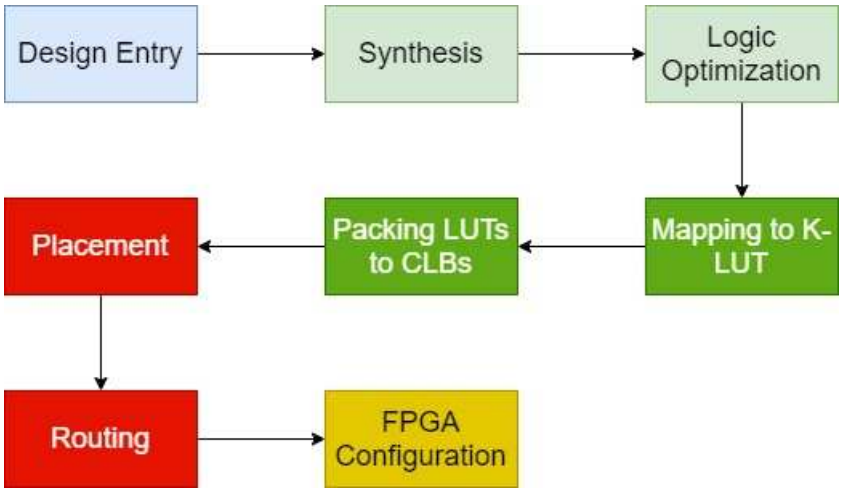


Figure 2.1: FPGA CAD Flow

1. Synthesis: This is the first step in the CAD flow. During this step, the CAD tool uses the description of the circuit as an HDL file and creates a netlist consisting of nodes and nets. Each node in the netlist represents a resource on the FPGA, while each net represents the connection between the nodes. During this step, various optimization techniques are used to simplify the logic, and to reduce the number

of gates and the complexity of the circuit. In the FPGA, all the logic gates are translated to look-up tables (LUT) that are later packed together to form basic-logic elements (BLEs) and configurable logic blocks (CLBs).

2. Placement: This step is performed after the synthesis of the design. During this step, the netlist is mapped onto actual resources on the FPGA. The main difficulty of the placement problem is the constraints that need to be satisfied. All the resources should be mapped to sites of their own type on the FPGA, the wire length should be minimized to fit the FPGA design, and all the timing constraints, including the target frequency and timing closure, should be met. In large designs, there are a large number of components that need to be placed under these constraints, and this has made placement an NP-hard problem.

3. Routing: This is the last step in the FPGA CAD flow. Routing involves mapping nets to actual wire segments and switches in the FPGA. FPGA architecture consists of prefabricated routing channels and switches that can be configured to connect components on the FPGA. Due to the limited number of available switches and wire segments, as well as the size of the design, routing can take several hours or days for large designs. The most important point is that the routing stage is heavily dependant on the quality of placement since it defines the location of the components on the FPGA.

2.2 Analytical Placement

Analytical placement is a subset of placement techniques that minimizes an objective, such as circuit delay, using mathematical techniques. GPlace is an analytical placement tool that performs placement on Xilinx Ultrascale FPGAs. Placement is performed on a flat level, meaning that logic elements are not packed into CLBs and are free to move during different stages to improve the quality of the results. GPlace has two different flows that can either minimize the estimated wiring requirement (wire-length driven) or minimize the critical path delay (time-driven). Both methods attempt to reduce the congestion through the FPGA device.

2.2.1 GPlace3.0

This is the wire-length driven flow of the GPlace. Figure 2.2 shows the overview of this flow that places a benchmark through three sequential phases. The input to the placer is a file that describes the architecture of the target FPGA and a netlist file. This netlist file includes all the cells used in the design, such as LUTs, FFs, DSPs, as well as the nets that describe the connectivity of the cells. The placer then outputs a placement file that contains the mapping of the design on the FPGA with respect to all of the constraints of the architecture. The role of each phase is as follows:

Phase 1: This phase attempts to reduce the wire length required to connect the components to each other. An initial pre-placement is performed to place the cells at the average position of the inputs and outputs for each cell. Experiments have shown that the quality of the results in the late stages of GPlace is enhanced by this pre-placement. Several iterations of the global placement are then performed to obtain a placement with

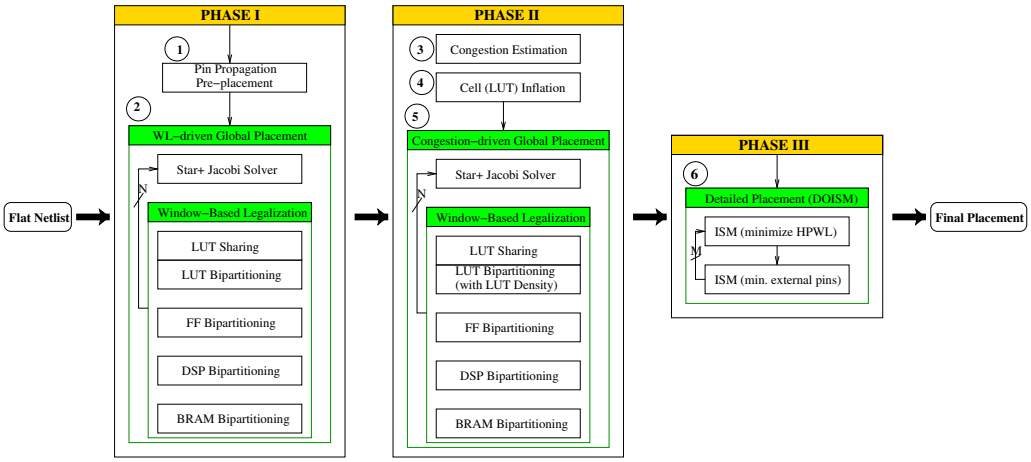


Figure 2.2: GPlace3.0 flow

a minimized wire length. At each iteration of the global placement, the star+ [13] net model seeks to reduce the wire length. At this stage, the cells are placed at sites that minimize the wirelength; these sites may, however, overlap. A window-based bipartitioning method is then used to legalize the placement. The goal of the legalizer is to remove the overlaps and ensure that the cells are placed in valid sites, while maintaining the minimum displacement of the initial solution.

Phase 2: The solution from the previous phase does not take congestion into account. This step therefore aims to reduce the congestion of the placement. First, a global router is used to estimate the congestion through all the switches used in the design. The size of the LUTs in the congested areas are then increased virtually to ensure that the cells in these areas are spread out. The amount of increase depends on the level of congestion in the area; the LUTs in the more congested areas are inflated more. In less congested areas, the size of the LUTs might remain the same or even be reduced. In this phase, a congestion-driven global placement is performed. First, the star+ solver optimizes the

wirelength; the congestion is then taken into account during legalization.

Phase 3: This phase is responsible for a detailed placement. The goal of a detailed placement is to further improve the wire length and routability by performing local optimizations. This phase uses dual-objective independent set matching that can alternate between the optimization of wire length and routability. DOISM selects cells that do not share nets in a window. A minimum cost matching then pairs each cell with a valid location to minimize the objective function.

2.2.2 GPlace4.0

This is the timing-driven flow [14] of Gplace that is built upon Gplace3.0; timing optimizations are incorporated in all the stages. First, the equations in the global placement (Phases 1 and 2) are modified to optimize both the wire length and the critical path delay. Then Phase 3 is modified using timing equations to improve wire length, routability and timing. The timing cost of each net is calculated using a timing graph. A timing graph is created in which each circuit element (i.e., LUT, FF, I/O, DSP, and BRAM) is represented by several timing nodes (i.e., t-nodes) that correspond to the input and output pins. Source and sink t-nodes represent the start and the end of the timing paths, and are added to each circuit element, which is driven by a clock. Timing edges (i.e., t-edges) are created to connect the t-nodes and represent both the internal connections between the pins of the circuit element and the nets that exist between the elements. Timing edges store the delay between the t-nodes. A timing analysis is run several times during the placement. The timing graph is traversed from the source t-nodes to the sink t-nodes,

thus setting the arrival time for each t-node with Eqn. 2.1:

$$t_{arrival}(sl) = \max_{i \in fanin(sl)} (t_{arrival(i)} + delay(sl, i)) \tag{2.1}$$

where *sl* is the net driver and *i* is the sink. A second traversal backwards through the graph sets the required time for each t-node as shown by Eqn. 2.2:

$$t_{required}(sl) = \min_{i \in fanout(sl)} (t_{required(i)} - delay(sl, i)) \tag{2.2}$$

For each t-edge, the slack is calculated by Eqn. 2.3:

$$slack(sl, i) = t_{required}(sl) - t_{arrival}(i) - delay(sl, i) \tag{2.3}$$

The delay function finds the delay between two t-nodes from a lookup table, which corresponds to the amount of delay that can be added to the t-edge before it becomes critical. Eqn. 2.4 is used to calculate the criticality of each t-edge:

$$crit(sl, i) = \left(1 - \frac{slack(sl, i)}{D_{max}}\right) \tag{2.4}$$

where D_{max} is the maximum delay in the circuit or a target delay value. Finally, the timing cost is calculated for each t-edge by Eqn. 2.5:

$$timing\_cost_{t-edge}(sl, i) = delay(sl, i) \cdot crit(sl, i) \tag{2.5}$$

Finally the overall objective function of the timing driven analytic solver is given by

Eqn. 2.6:

$$f(x) = \sum_{\forall l \in circuit} \lambda \times S_l + (1 - \lambda) \times T_l \tag{2.6}$$

λ is a trade-off parameter in the range of $[0, 1]$, which balances the objectives of both the wirelength and the timing.

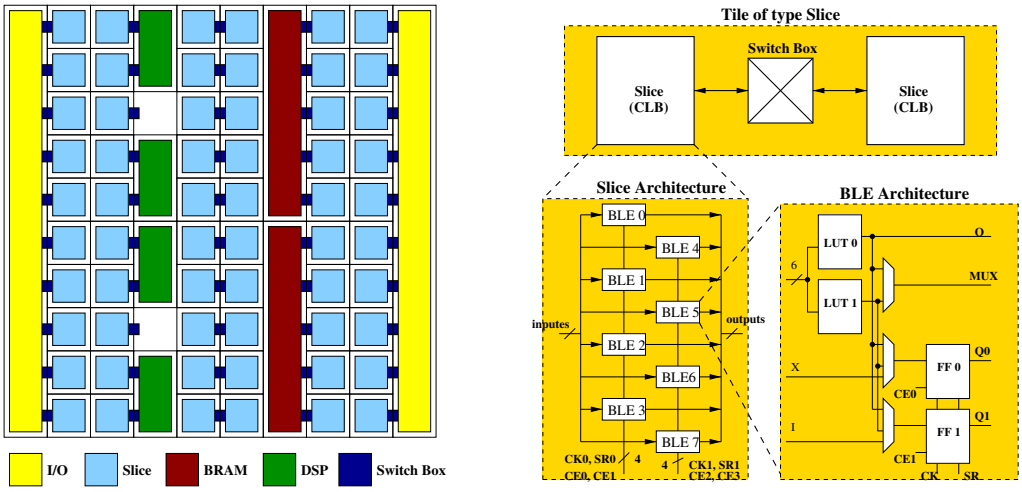
2.3 UltraScale FPGA Architecture

In this thesis, a Modern Xilinx Ultrascale VU095 FPGA device is targeted. This device has a heterogeneous architecture, which means it contains a mixture of logic blocks. These blocks can consist of lookup tables (LUTs), flip-flops (FFs), block random-access memory (BRAM), digital-signal processing (DSP) blocks, and input/outputs (I/Os). Figure 2.3a shows the layout of the FPGA. DSPs, BRAMs, and I/Os can be placed on the sites of their own type, while LUTs and FFs are placed in the slices.

In this device, each slice contains only one CLB. The architecture of each slice is shown in Figure 2.3b. Each slice can have up to eight basic logic elements (BLEs). There are two clock and two set/reset signals. Half of the BLEs use the same clock and set/rest signal, while the other half use the second pair. Each half is then divided into two smaller subgroups that must each use the same clock enable (CE). BLEs consist of two FFs and a LUT that can be configured to implement one function with six inputs or two different functions with up to five distinct inputs. The control set constraints, combined with the limited FPGA resources, make the placement problem very challenging.

Combined with the resources, there are programmable switches, and horizontal and vertical prefabricated wire segments with different lengths. These segments can be con-

figured to connect different components on the FPGA device.



(a) UltraScale Layout.

(b) Tile Architecture of Slice Type.

Figure 2.3: Xilinx UltraScale Architecture.

2.4 Benchmarks

In this thesis, an ML model is used to predict the maximum frequency of operation during placement. The accuracy and generalization of the model are highly dependent on the quality and quantity of the benchmarks. With this in mind, the 372 benchmarks available from [15] were used to target the modern ultra-scale V095 FPGA device. This data set consists of 12 original benchmarks from ISPD 2016 [16] as well as 30 variants for each one that was created using the netlist generation tool. Table 2.1 shows the key circuit features for the 12 original benchmarks. As well, Table 2.2 shows the range of the key features for the 372 benchmarks. The size of these benchmarks covers a wide

range. The size varies from 0.1 to 1.1 million gates, which makes the placement problem challenging.

Benchmark	#LUTs	#FF	#BRAM	#DSP	#CSet[a]	#IO	R.E[b]
FPGA-1	49K	55K	0	0	12	150	0.4
FPGA-2	98K	74K	100	100	121	150	0.4
FPGA-3	245K	170K	600	500	1281	400	0.6
FPGA-4	245K	172K	600	500	1281	400	0.7
FPGA-5	246K	174K	600	500	1281	400	0.8
FPGA-6	345K	352K	1000	600	2541	600	0.6
FPGA-7	344K	357K	1000	600	2541	600	0.7
FPGA-8	485K	216K	600	500	1281	400	0.7
FPGA-9	486K	366K	1000	600	2541	600	0.7
FPGA-10	346K	600K	1000	600	2541	600	0.6
FPGA-11	467K	363K	1000	400	2091	600	0.7
FPGA-12	488K	602K	600	500	1281	400	0.6

Table 2.1: ISPD 2016 Placement Contest Benchmark Statistics

[a]#CSet: Combination of reset and control-enable signals in the benchmark

[b]Rent Exponent (RE)

#LUTs	#FF	#BRAM	#DSP	#CSet	#IO	R.E
44K-518K	52K-603K	0-1035	0-620	11-2684	150-600	0.4-0.8

Table 2.2: Range of Key Circuit Features

2.5 Maximum Frequency of Operation

The maximum frequency of operation is the highest clock rate at which a digital circuit can work while correctly performing all the operations without any glitches or hazards.

F_{max} is important because when it is higher, it means the circuit can work with a higher clock rate and produce the desired output in less time. In a digital design, several modules are considered to be a sink or a source of the path (for example, inputs and outputs of the FFs). As this is based on the actual physical location of each module, the delay in this path can vary from one path to another. During each clock cycle, there should be enough time for the data to be transferred and captured from the source to the sink. This time consists of three main parts:

- Setup time
- Hold time
- Path delay

Setup time is the time in which the input data signals should be stable before the active clock edge occurs. Hold time is the time in which the input data signals should be stable after the active clock edge occurs. Path delay is the time that it takes for the data to travel from the source to the sink. Setup and hold times must be satisfied to ensure that the circuit is working correctly. The F_{max} should be determined by going through all the paths in the design and finding the one with the maximum delay. This delay is called the critical path. When the timing requirements of the critical path are met, the circuit is working correctly. The following equations should then be satisfied:

$$T_{PD} + T_{Setup} \leq T_{Clk} \tag{2.7}$$

$$T_{Hold} \leq T_{PD} \tag{2.8}$$

2.6 Machine Learning

Machine learning is a branch of artificial intelligence that provides the ability to learn from data and predict the results by finding the hidden relationship between inputs and outputs. Machine learning has become very popular lately due to the availability of large datasets, the increases in memory, the reductions in computational limits, and the high run times of traditional algorithms. There are various benefits to using ML models in different applications: they are fast and can learn the patterns and trends in data that might not be apparent to humans; they do not need any human interaction and hard coding, which can save a lot of time in the implementation phase; and they keep improving through the availability of new datasets.

One of the most important challenges in ML is error generalization. During the training stage, the model learns from the training data. The goal is to have good predictions for the unseen data. A model would be considered over-fitted when it performs well using the training data but performs poorly on the test data. This means that the model has memorized the training samples. In comparison, the model would be under-fitted if it has not seen enough data to learn from it. There should always be a balance between these two extremes to improve the performance of the model.

ML models can be categorized into four subgroups based on their applications and method of learning.

- Supervised learning: In supervised learning, the model learns by observing examples. An algorithm with a desired set of inputs and outputs is provided. The model then tries to find patterns in the data and predict new outputs. These outputs are corrected by the supervisor. This process then continues until a high degree of

accuracy is obtained.

- Unsupervised learning: In unsupervised learning, there is no correct output or interaction with the supervisor. The model is provided with a large set of unlabeled data. It then tries to find patterns and learn the structure of the data. The output of this model can be a more organized looking cluster or arrangement of data.

- Semi-supervised: In this approach, the model is provided with a data set consisting of labeled and unlabeled data. The model falls between supervised and unsupervised learning during training and tries to label the unlabeled data.

- Reinforcement learning: In this method, the model is provided with a set of actions and rules, as well as both stated and end values. The model then tries to explore the solution space by monitoring and evaluating different options to determine which one is optimal. It learns from the past and adapts its decisions in response to the current state to achieve the best possible results.

ML models can also be categorized into three subgroups based on their application.

- Regression: Regression is the task of predicting a real continuous value based on the inputs. Predicting the price of a house based on inflation and location is an example of a regression problem in real life.

- Classification: Classification is the task of determining the class label to which each observation belongs. Labeling emails as either spam or not spam is an example of classification in real life.

- Clustering: Clustering is the task of dividing data into subgroups such that the data in each group are close to each other. Clustering the viewers in streaming services is an example of clustering algorithms in real life.

2.6.1 Predictive Models

In this thesis, mostly supervised learning models were used for regression problems and clustering algorithms. The models that were used in this thesis are defined in detail below:

- Linear regression: This is a supervised ML model that performs the regression task. It assumes that there is a linear relationship between the inputs and the output and tries to learn the weight matrix for the inputs to reduce its cost function. This technique is faster than the other models and is effective when the relationships between the inputs are simple. It usually suffers from a lack of accuracy since many problems in the real world are not linear. As well, the outliers can highly affect the model, which might not be desirable in some applications.

- SVM: Support vector machines (SVM) are a form of a machine learning model that can perform both classification and regression tasks. SVM attempts to predict outputs by creating a line or a hyperplane to classify data. This method works best when there is a clear margin between the classes and when the number of data dimensions is higher than the samples. This model is also very memory efficient. Despite its advantages, this model contains several drawbacks. It has a high training time and cannot perform well on large datasets. As well, the quality of the predictions is low when there is an overlap between the classes.

- Decision tree: A decision tree (DT) attempts to predict the results by transforming the data into a representation of a tree. This helps to both evaluate and choose between the different courses of action. The main advantage of a DT is that it can consider all the possible outcomes of each decision thus leading to a conclusion. As well, a DT does not require normalization or scaling, and the missing data cannot affect the model. The disadvantages of this model include its high degree of complexity and its training time. This model is also very dependant on the dataset, and a small change can affect the structure of the optimal tree.

- Random forest: This is an ensemble ML technique that tries to create a prediction model using weak learners. The idea behind this model is that a large number of uncorrelated models operates as a committee and can outperform each of the individuals. This model is based on the bagging method, which uses multiple decision trees to increase the prediction performance. In bagging, weak learners are trained in parallel on random subsamples of data. This method aims to decrease variance and avoid over-fitting. This model is robust to outliers and missing data, and works well on large datasets. The main drawbacks of this model are its high complexity, as well as its training and inference times.

- Gradient boost: This is another subgroup of ensembles that is based on combining the weak learners. The training proceeds iteratively, adding new trees that predict the residuals or errors of the prior trees that are then combined with previous trees to make the final prediction. The main difference between XGB and RF is the training process. In RF, the training process is performed in parallel while XGB is sequential. This model aims to reduce biases and variances. XGB can handle the

missing data, and there is no preprocessing required. The main drawback of this model is the high timing costs. The timing cost is even higher than that of other ensembles like RF due to its sequential nature.

- Stacking model: This model uses a meta-learning algorithm to combine the predictions from multiple models and create a meta-model with fewer biases and variances. This model works best when there are multiple models that have strength in different subsets of data. The main problem with this model is the high training time.

2.6.2 Feature selection

Feature selection is an important step in developing a machine learning model as it tries to reduce the number of inputs to the model. This reduction can be important with regards to several aspects. First, with fewer inputs, the complexity of the model is reduced, and it is thus likely to have less training and inference time. Second, many features might have a low correlation to the label or be uninformative, and removing them from the model can increase its accuracy. In this thesis, sequential backward selection (SBS) was used as the feature selection technique. This technique begins with the full set of candidate features. This set is then iterated repeatedly, and with each iteration, the least-important feature is removed. This process continues until the MSE of the predictions starts to increase.

2.6.3 Hyperparameter tuning

Hyperparameter tuning is another important step in training an ML model. In ML, a hyperparameter is a parameter that can control the learning process and can have a great effect on the performance of the model. In this thesis, two hyperparameter tuning methods were used: random search and grid search. In random search, a grid of parameters is set and random combinations are used to train and test the model. In comparison, grid search uses all valid combinations and tests them. In this work, the search range was narrowed down by first using a random search with a wide range of hyperparameters. A grid search was then performed to find the best set of hyperparameters for each model.

2.6.4 Performance Metrics

An important element in ML is how to evaluate the performance of the model. Since each of the metrics can show a different characteristic of the model, multiple metrics can be used to help find all the pros and cons of a model. This thesis employed five different metrics that are widely used for regression problems. In all the metrics, y_i represents the actual value of point i, while $\hat{y}_i$ is the estimated value of the same point. The five metrics are as follows:

- Accuracy: This gives a measure of how well the model replicates the observed outcomes, and is calculated using Equation 2.9.

$$R^2 = 1 - \frac{\sum_{i=1}^{N}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{N}(y_i - \bar{y})^2} \tag{2.9}$$

- Mean square error (MSE): This is used to check how close the estimates or fore-

casts are to the actual values, and is calculated using Equation 2.10. This metric squares the error, so it amplifies the large error in the prediction and the average error of the model. However, the units for MSE are squared units, which can be confusing.

$$MSE = \frac{1}{N}\sum_{i=1}^{N}\left(y_i - \hat{y}_i\right)^2 \tag{2.10}$$

- Root mean square error (RMSE): RMSE gives a relatively high weight to large errors. This means that RMSE is most useful when large errors are undesirable, and is calculated using Equation 2.11. It also solves the problem of using the squared units of the MSE.

$$RMSE = \sqrt{\frac{1}{N}\sum_{i=1}^{N}\left(y_i - \hat{y}_i\right)^2} \tag{2.11}$$

- Mean absolute error (MAE): This measures the average magnitude of the errors in a set of forecasts. This is a linear score, which means that all the individual differences are weighted equally in the average. It is calculated using Equation 2.12.

$$MAE = \frac{1}{N}\sum_{i=1}^{N} \mid y_i - \hat{y}_i \mid \tag{2.12}$$

- Mean absolute percentage error (MAPE): This permits one to compare the forecast accuracy between differently scaled data. It is calculated using Equation 2.13. In this work, the scale of the predictions differs from one circuit to another; therefore this is a useful metric for this work.

$$MAPE = \frac{1}{N} \sum_{i=1}^{N} \frac{1}{y_i} \mid y_i - \hat{y}_i \mid \tag{2.13}$$

2.7 Summary

This chapter introduced FPGAs, CAD flows, and placement problems and challenges. The architecture of the target FPGA device and its constraints were then presented. The GPlace analytical tool and its timing-driven flow were introduced along with the concepts of F_{Max} in digital circuits. Finally, machine learning was discussed. This included the steps that need to be performed in order to train and test a model. The ML models that were used in this thesis were also introduced.

Chapter 3

Literature Review

This chapter covers some of the previous works in the literature that are related to this thesis. Section 3.1 addresses works related to the applications of ML in CAD flow as well as those related to the probes during placement. Section 3.2 discusses previous works on predicting frequency in the ASIC design, while Section 3.3 focuses on the frequency prediction task in FPGAs.

3.1 Machine Learning in CAD Flow

Many works have addressed the use of machine learning to deal with the placement problems for the FPGA. These works have used ML models to predict issues in the flow and utilised this information to guide the placement engine. The main goal during the placement is routability, and this parameter can only be determined after the routing stage has been performed. The authors in [5] proposed a deep-learning model called DLRoute, which is based on a convolutional neural network that can determine if a placement

file is routable or not. These predictions can be used as probes during placement to reduce the runtime and increase the quality of the placement by dynamically adjusting the optimization strategy of the placer. Using DLRoute can save from 4.71% to 82.1% in total place and route time and avoid the production of unroutable placements. Later the work in [17] further improved the results of DLRoute by using traditional ML models instead of CNN. This model was able to produce results that were 1% more accurate and had fewer false positives. In addition, the inference and training time for this model was a fraction of that of DLroute, which makes it a better candidate to be used as a probe during placement. The work in [18] focused on the gap between the global router and the detailed router in the FPGA placement. This gap has risen due to the increasing complexity and size of the designs. This paper used a method based on machine learning techniques to predict detailed routing violations with extracting relative features.

Another important parameter to be considered during placement is congestion. Authors in [6] proposed a machine learning model based on linear regression. This model used four features that described the connectivity and routing demands on a bounding box to estimate the congestion during the placement for each switch. These predictions could be used later in congestion management to reduce congestion and increase the routability of the design. The experiments showed that by using MLCong, the runtime of the router was 19% lower. The work in [19] presented a novel deep-learning technique for congestion estimation called DLCong. This framework used a feature map instead of four individual features and could achieve an improvement of 9% in comparison to MLCong. This framework also scaled better when the congestion increased.

3.2 Frequency Prediction in ASICs

The work in [20] tried to fill the gap in the timing analysis of a design before and after routing. Traditionally, an overly pessimistic prediction would be used to ensure that no timing violations would occur after routing; however, this method leads to a waste in power and area. The other solution is to iterate back to the placement of the cells that the desired timing cannot achieve after routing. This method, however, can drastically increase the run time. The study proposed a framework based on the features in the net-level of the design to predict the timing analysis. Features, such as the distance of the source and the sink of the net, their capacitance, and the signal transition time needed to predict the delay of the nets, were used. The results indicated that a 97% correlation with the post routing results could be obtained. When compared to a commercial timing estimation, this method had 66% fewer false positives.

The work in [21] targeted the floorplan stage. This stage is mostly spent on the generation and evaluation of solutions. Each solution must go through placement and clock tree synthesis routing. As a result, it can take days to complete. The framework proposed in this study predicted the post-route slack of the SRAMs using the features available from the netlist and floorplan stage, and gave an accurate estimation of each solution. A mean absolute error of 23.03 ps could be achieved, and the evaluation time could be reduced from 8 hours to 60 seconds. The main drawback of this work was that it skipped the effect of the middle stages; therefore, a good floorplan with a weak placement might not satisfy the timing constraints.

The work in [22] considered the timing failures of embedded memories during the floorplan stage. Embedded memories are very important since they occupy substantial

die area. This can create placement and routing blockage, and can determine the failure of a complex (SoC). The authors proposed a machine learning framework to predict embedded memory timing failure using features that involved netlist, timing constraints, and the location of memory cells in the design. A degree of precision and recall of 93.3% and 95% were achieved, respectively. This early prediction of timing failures could reduce the design costs and turnaround time.

The work in [23] investigated the problem of the variability between the parameters and the environmental conditions. This model included temperature as a new feature in order to predict the timing analysis under different conditions. A median error of less than 5% was reached. The case studies showed that the access time of the caches could vary up to 2.03X at high temperatures; this ML framework could therefore be very useful in these applications.

The work in [24] targeted the frequency variation in circuits working in non-nominal voltage. Since previous methods, such as STA, did not take the voltage into account, they were unable to support timing for low voltage or sub-threshold voltage designs. This study proposed a framework that could estimate the timing of a design based on its working voltage with a maximum error of 7.9%. The proposed method was tested on a RISK core with a voltage that varied from 0.5 to 0.9.

3.3 Frequency Prediction in FPGAs

The frequency prediction task in FPGAs is quite new. There are works that have tried to predict the frequency at different stages of the design flow, but to the best of the authors' knowledge, this thesis is the first time that the frequency prediction task has

been performed during placement using ML.

The work in [25] tried to increase the accuracy of the static timing analysis and the timing slacks of the design. Graph-based analysis (GBA) and path-based analysis (PBA) are the two techniques that have been used to estimate the timing slacks. Although PBA is more accurate, it requires four times as much runtime to calculate. The authors of this study used the GBA to predict the PBA using machine learning models. The main problem with this work is the relatively high computational time required for the features. This could be problematic when in frequent use during the process. Although this work could improve the STA prediction, there is still a gap between the STA and the actual frequency. The post-routing frequency is affected by many dynamic parameters that STA does not consider.

The work in [26] considered the problem of HLS tools, which have a timing report that deviates from the post-implementation results. This work proposed a framework called a pyramid that used machine learning to predict resource usage and the maximum achievable frequency of an HLS design. First, an automated hardware optimization tool was used to find the maximum frequency for each design. The model then used a set of features to describe performance, resources, logic, and arithmetic operations and memories that were available from the HLS report. Results indicated that the pyramid could obtain an accuracy of 95% when predicting throughput and throughput-to-area.

The work in [27] presented a framework that modeled FPGA digital signal processing (DSP) applications at an abstract level. This framework produced predictions of parameters, such as clock frequency, resource utilization, and latency. Based on several benchmarks, the study showed an average clock frequency prediction error of 3.6% with a worst-case error of 20.4% when compared to the best existing high-level predic-

tion methods. The study also had an average error of 13.9% with a worst-case error of 48.2% in resource utilization predictions. These predictions would allow an accurate exploration of design space without the requirement of coding in a hardware description language (HDL). This would reduce the total design runtime.

Authors in [28] proposed an estimator for finding the frequency of a design and the area after the place and the route stages from a description of the design as an algorithm in MATLAB. The study showed that the frequency estimations were within 13% of the actual synthesized frequency. As well, the total number of CLBs predicted was within 16% of the actual number of the CLB. This accurate and fast framework could be used to perform rapid design exploration.

3.4 Summary

This chapter introduced previous works that have targeted the challenges that occur during placement when using ML approaches. These challenges include routability predictions and congestion management. Some of the previous works regarding frequency prediction in FPGAs and ASIC were presented along with the advantages and disadvantages of each system.

Chapter 4

Methodology and Proposed Framework

In this chapter, the proposed methodology for this thesis is introduced. As described in Chapter 1, STA has been widely used as the delay model in most timing-driven placement flows. However, the accuracy of STA for the F_{Max} is low in most cases. This can reduce the quality of the placement or cause the placement to not satisfy the timing closure after routing. This thesis developed a machine learning framework that can predict the F_{max} at each stage of placement. These predictions can later be used as probes to guide the placement strategy. Section 4.1 introduces the ML framework for predicting F_{max}, including aspects such as data preparation, training, testing, and feature selection. Section 4.2 presents the deployment stage and the integration of the F_{max} predictor in GPlace. Finally, Section 4.3 presents the case studies of the ML framework in GPlace, and how this probe can improve the placement results.

4.1 ML framework

Figure 4.1 shows the proposed ML framework for predicting F_{max} during placement. The steps are discussed in detail.

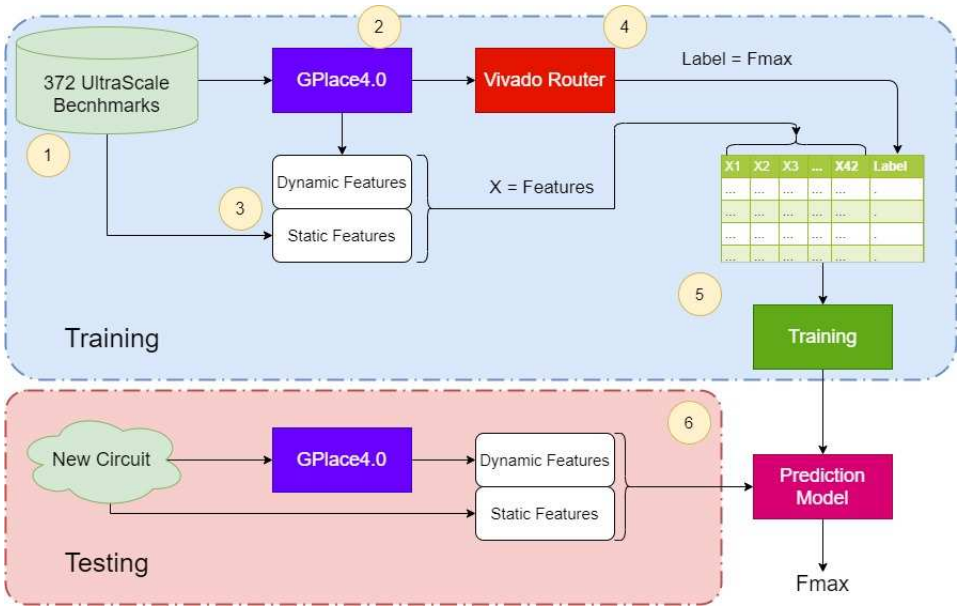


Figure 4.1: Proposed Framework

4.1.1 Benchmarks

The quality and quantity of the dataset are crucial if the accuracy and generalizations of this model are to be increased. This step uses the 372 benchmarks described in Section 2.4. These benchmarks have been used in several works regarding placement and routing, and consist of a wide range of circuits with different features that make them suitable candidates for this model. In order to gain a better understanding of these benchmarks, a PCA and a K-means algorithm were used to cluster them based on the key circuit

features. Figure 4.2 shows the results for the K-means algorithm. The Y-axis shows the cluster number, and the X-axis is the number of elements from each group that belong to it. A silhouette score and an elbow method were used to find the optimum number of clusters. Both methods verified that the best K is 7. The figure shows that some groups, such as 3, 4 and 5 or 6, 7 and 8, are clustered together.

Figure 4.3 shows the results of the PCA, in which the dataset was reduced to a 2-dimensional space in order to be visualized. Both algorithms verified that the dataset has several subgroups. Prior knowledge indicates that although some groups, such as 3, 4 and 5, are identical to each other in terms of the key features, they behave differently with regards to congestion, routability, and frequency. At the same time, other groups that are far apart, such as 10 and 5, have similar frequencies. These experiments showed that predicting frequency can be a challenging task as frequency is very sensitive to many parameters. Accurate feature engineering must be performed to capture all the parameters of the benchmarks so that the model learns the behaviour of the circuit and the router.

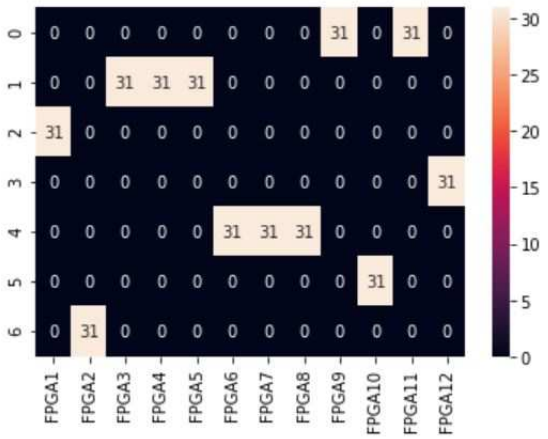


Figure 4.2: K-means on 372 Benchmarks

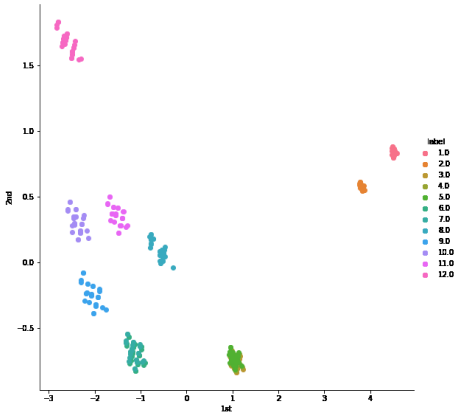


Figure 4.3: PCA on 372 Benchmarks

4.1.2 Placer

In this step, each of the benchmarks were placed using Gplace with a timing-driven flow. To increase the quantity of the dataset, the source code for GPlace4.0 was changed so that after each iteration during the global and detailed placement, intermediate placement files were captured. These are legal placement files that can be used to create a larger dataset. In the end, it was possible to produce 3618 routable placement files from the original benchmarks.

4.1.3 Feature Engineering

Another important element that is required to accurately predict the maximum frequency of a design is the features used for the ML model. For this purpose, two sets of features were used:

- Static features

- Dynamic features

Static features are extracted from the circuit netlist and can show different parameters of the benchmarks, such as the number of different modules or their connectivity. These features are calculated before placement and can provide an estimation for the upper bound of the frequency. Since the frequency of a circuit depends on various parameters, and static features cannot provide an accurate prediction, dynamic features are used. Dynamic features are extracted from the placer during placement, and capture parameters, such as congestion, timing analysis, and routing demands, which are important when predicting the frequency.

4.1.3.1 Static Features

Resource Features: Table 4.1 shows a list of the resource features. These features capture the resource usage of the FPGA. Features 1 to 11 show the number of LUTs with sizes from 2 to 6, as well as the FFs, DSP, RAM, I/O pins, and control sets, respectively. These features are important because they are correlated with the FPGA utilization during placement and can affect the routing of the design. Each of these resources has a specific site on the FPGA and can affect the routing demand from one site to another. This makes them an important factor in determining F_{max}. These features are properties of the circuit, and only need to be read from the netlist, which takes less than a second. GPlace reads this data at the start of the placement so that if the data is required later, there is no computation time.

Connectivity Features: Table 4.2 shows the list of connectivity features. These features capture the connectivity of the modules in the netlist. Features 12 to 17 show the percentage of nets with 2, 3, 4, 5 to 10, 10 to 30, and 30+ pins. Features 18 to 19 show the total

Number	Feature Name	Calculation	Computation Time (s)
1-8	Cell Type Counts	$\#Cell_i \in NL$	< 1
		$i = \{LUT2..6, FF, DSP, BRAM\}$	< 1
9	LUTS	$\#LUT2 + \#LUT3 + \#LUT4 + \#LUT5 + \#LUT6$	< 1
10	IO	$\#I/Os \in NL$	< 1
11	CSET	$\#CSETs \in NL$	< 1

Table 4.1: List of Resource Features

Note: NL represents the circuit netlist.

number of pins and nets in the netlist. These features capture the fan-out of the modules; for example, a 30-pin net shows that the output of a module is connected to 29 other pins. Fan-out can increase the delay in a path, and this delay is highly correlated with F_{max}. As well, the number of pins and nets in the netlist can show the routing demand and the connectivity of modules across the FPGA. These features are properties of the circuit, and only need to be read from the netlist, which takes less than a second. GPlace reads this data at the start of the placement, so that if the data is required later, there is no computation time.

Number	Feature Name	Calculation	Computation Time (s)
12-17	Net Type Counts	$\frac{\#i - pin\ nets \in NL}{\#nets \in NL}$	< 1
		$i = \{2, 3, 4, [5-10], [11-30], 30+\}$	< 1
18	PINS	$\#pins \in NL$	< 1
19	NETS	$\#net \in NL$	< 1

Table 4.2: List of Connectivity Features

Note: NL represents the circuit netlist.

Longest Path Features: Table 4.3 shows a list of the longest path features. These features correspond to the longest path in the netlist for all sink nodes. The netlist used in this thesis consists of different modules that could be the sink or the source of a path. For example, the input of a FF is a sink, and its output is a source. For each sink node

in the netlist, one finds all the source nodes that end in it. Then the length of the path between the source and the sink is calculated by assuming unit delays for each module. In the end, the longest path for each sink node can be found. Features 21 and 22 show the mean and the standard deviation (STD) of the longest path for all the sink nodes. Feature 20 shows the longest path in the netlist. Features 23 and 24 show the percentage of paths within 90 to 100% and 80-90% of the longest path. These features are important because there is a chance that the critical path in the final placement correlates with the longest path in the netlist. As well, when the longest path for a design has a high mean, the routing demand is likely to be higher. In order to compute these features, a graph of the circuit must be created. This takes less than four seconds depending on the size of the benchmark. These features can be computed before the start of the placement, and can be used repeatedly afterwards.

Number	Feature Name	Calculation	Computation Time
20	LP	$max(lp_{sinks})$	< 4
21	LPMEAN	$mean(lp_{sinks})$	< 4
22	LPSTD	$std(lp_{sinks})$	< 4
23	LP10	$\frac{\#sinknodes \in NL \mid lp \in [90\%LP, LP]}{\#sinknodes \in NL}$	< 4
24	LP20	$\frac{\#sinknodes \in NL \mid lp \in [80\%LP, 90\%LP)}{\#sinknodes \in NL}$	< 4

Table 4.3: List of Longest Path Features

Note: lp represents the longest path for a sink node and NL represents the circuit netlist.

4.1.3.2 Dynamic Features

Congestion Features: Table 4.4 shows a list of congestion features. These features characterize the congestion in the placement, and they are produced using the MLCong [6] congestion estimator. Features 25 to 26 show the mean and the STD of the congestion

among the switches that are used. Congestion can affect the routing of the design. Since the critical path is determined during routing, congestion features can play an important role in accurately determining the F_{max}. The MLCong is a machine learning model that estimates congestion, and the total feature extraction and inference time would be less than four seconds depending on the benchmark.

Number	Feature Name	Calculation	Computation Time (s)
25	CONGMEAN	$mean(Cong_{sw} \mid sw \in S_d)$	< 4
26	CONGSTDEV	$std(Cong_{sw} \mid sw \in S_d)$	< 4

Table 4.4: List of Congestion Features

Note: S_d represents switches used in the design.

Net-based Features: Table 4.5 shows a list of net-based features. These features capture the distribution of the nets in the placement and the routing network. Features 27 and 28 estimate the mean and the STD of the routing demand of each net by estimating the wire length of the net within a bounding box. Features 29 to 32 represent the absolute number of net cuts in a 5*5 and 9*9 window around a site. Features 33 and 34 represent the distribution of pins in the placement, which are related to switch congestion. The total computation time for these features is less than four seconds. These features are presently used in other ML frameworks that have been currently implemented in GPlace, such as MLCong or MLRoute; they are therefore computed in the current flow.

FPGA Utilization Features: Table 4.6 shows a list of the FPGA utilization features. These features capture the utilization of the FPGA. Features 35 to 36 are the half-perimeter wire length and the switch utilization of the FPGA. Wire length and switch utilization affect the routing stage, and they are important features in determining the

Number	Feature Name	Calculation	Computation Time
27	WLPAMEAN	$mean(WLPA_{sw} \mid sw \in S_d)$	< 4
28	WLPASTD	$std(WLPA_{sw} \mid sw \in S_d)$	< 4
29	NCPR5MEAN	$mean(NCPR_{W_{5\times5}(sw)} \mid sw \in S_d)$	< 4
30	NCPR5STD	$stdev(NCPR_{W_{5\times5}(sw)} \mid sw \in S_d)$	< 4
31	NCPR9MEAN	$mean(NCPR_{W_{9\times9}(sw)} \mid sw \in S_d)$	< 4
32	NCPR9STD	$stdev(NCPR_{W_{9\times9}(sw)} \mid sw \in S_d)$	< 4
33	PINMEAN	$mean(pins_{sw} \mid sw \in S_d)$	< 4
34	PINSTD	$std(pins_{sw} \mid sw \in S_d)$	< 4

Table 4.5: List of Net-based Features

Note: S_d represents switches used in the design, $W_{n\times n}(sw)$ represents an $n \times n$ window centered on sw.

F_{max}. The total computation time for these features is less than one second. These are properties of the current placement. They are computed repeatedly during the flow, so they are ready whenever they are needed.

Number	Feature Name	Calculation	Computation Time (s)
35	HPWL	$\sum_{net \in NL} HPWL_{net}$	< 1
36	SW	$\#sw \mid sw \in S_d$	< 1

Table 4.6: List of FPGA Utilization Features

Note: S_d represents switches used in the design.

Critical Path Features: Table 4.7 shows a list of the critical path features. These features show the characteristics of the critical path in the placement. The critical path is estimated using static timing analysis in [14]. Features 37 to 38 show the delay and the length of the critical path. Features 39 to 40 show the mean and the STD of the size of the nets in the critical path. Features 41 and 42 show the mean and the STD of the congestion for the switches that are in the critical path. F_{max} is calculated by using the actual critical path during routing. However, the critical path used is during the placement, and there

is no guarantee that it is the final critical path. By using other features and the estimated critical path, it is expected that F_{max} can be found accurately. The mean and the STD of the nets in the path show the fan-out of modules in the path. This can help predict the actual delay more accurately. As well, the congestion in the critical path can indicate the accuracy of the estimations in the final design since congestion in these areas can affect the routing. Locating the critical path takes less than three seconds, and all the other features are computed immediately afterwards.

Number	Feature Name	Calculation	Computation Time (s)
37	CPL	$\#module \mid module \in cp$	< 3
38	CPD	arrival time of last module in cp	< 3
39	NETMEAN	$mean(netsize \mid net \in cp)$	< 3
40	NETSTD	$std(netsize \mid net \in cp)$	< 3
41	CPCONGMEAN	$mean(Cong_{sw} \mid sw \in S_{cp})$	< 3
42	CPCONGSTD	$std(Cong_{sw} \mid sw \in S_{cp})$	< 3

Table 4.7: List of Timing Features

Note: cp represents the critical path, S_{cp} represents the switches used in the critical path.

4.1.4 Label Extraction

Since supervised learning was used, all of the data must be labeled. There were 3618 routable placements produced in Section 4.1.2. All of the placement files should then be routed with timing optimizations to find the operating frequency. However, Vivado does not give the maximum clock frequency for a circuit; it merely attempts to meet a target clock frequency. A binary search is thus required to find the maximum frequency, and this can be a prolonged process for the larger/more congested benchmarks. Alternatively, a very high clock frequency (e.g. a clock period set to 0.1 ns) could be used. The resulting

log file would report a negative slack value, which could be subtracted from 0.1 ns to find the clock period. The clock frequency could then be calculated. Figure 4.4 shows the frequency of each placement; different groups are separated and shown from Groups 1 to 12, respectively. The dataset ranges from high frequency (250 MHz) to low frequency (50 MHz) placements. As well, the frequency is more scattered in higher frequency groups, such as 1, 2, 3 and 8. Accurately predicting the F_{max} in these groups would be more challenging.

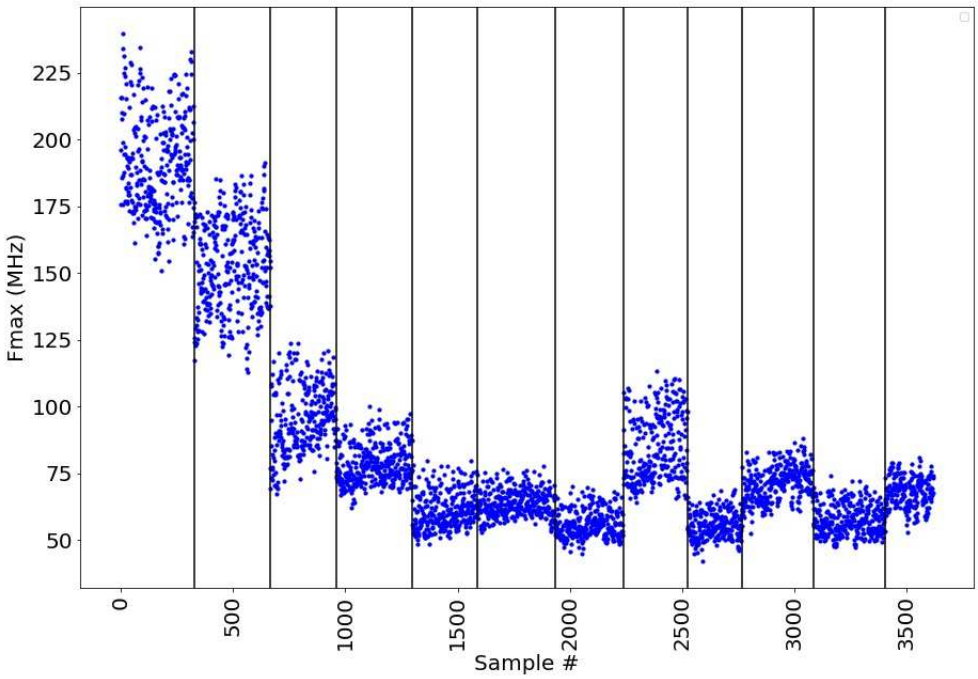


Figure 4.4: Frequency for all Placements

4.1.5 Training

The dataset was sampled randomly and 70% was assigned to be trained, 15% was assigned to the validation, and 15% was assigned to the testing set. The trained set was used for training, while the validation set was used to tune the hyperparameter and the feature selections. For the predictive model, six simple and ensemble modes were used to investigate the performance of each model in order to pick the best one. The simple models used in this thesis were:

- Linear Regression
- SVM
- Decision Tree

Ensembles used in this thesis were:

- Random Forest
- XGBoost
- Stacking Model

Before training, the dataset, which included the features, was standardized to zero mean and unit standard deviations. This was due to the sensitivity of some models to the size of the features,

4.1.5.1 Feature Selection

Feature selection is an important step in training the model. A large number of features can increase the timing costs and the time complexity of the model. However, uncor-

related features can reduce the accuracy and the quality of the predictions. Figure 4.5 shows the correlation heatmap of the dataset. The correlation between 1 and -1 shows a perfect positive and negative correlation, respectively, while 0 means no correlation. Some features, such as SW and CPD, are highly correlated with the label, while this value is small for features such as LP10 or NETMEAN. Due to this variation in the importance of each feature, it was necessary to use feature selection in the work. To find the best set of features, SBS was used as discussed in Section 2.6.2.

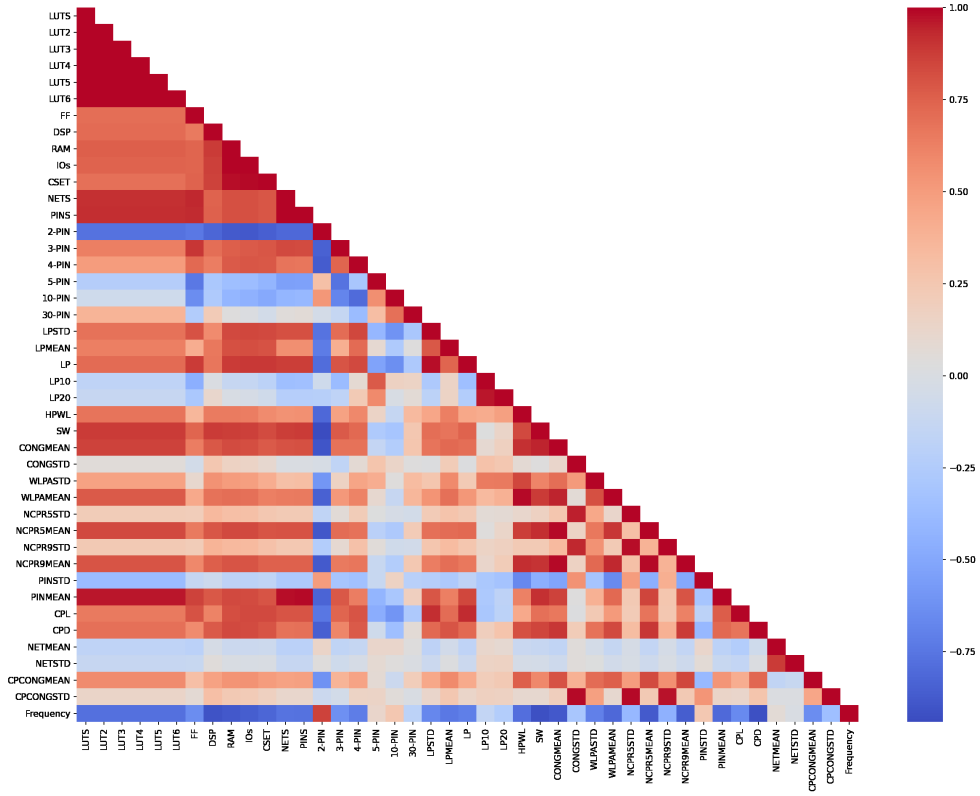


Figure 4.5: Dataset Correlation Heatmap

4.1.5.2 Hyperparameters Tuning

Hyperparameters are one of the important factors that affect the accuracy of the predictions. For this purpose, a validation set was used. First, the search range was narrowed down by using a random search with a wide range of hyperparameters. Then a grid search was performed to find the best set of hyperparameters for each model. Table 4.8 shows a list of the most important hyperparameters for each model along with the search range for each parameter.

Model	Hyperparameter	Range
LR	Learning Rate	0.1-0.5
SVM	Kernel	Radial Basis Function-Linear-Poly
	Regularization Param	0.1-30
	Kernel Coefficient	scale-auto
DT	Criterion	MSE-MAE
	Max Depth	5-50
	Min Samples Split	2-12
	Max Features	Sqrt-Auto
RF	Criterion	MSE-MAE
	Number of Estimators	20-100
	Min Samples Split	2-10
	Min Samples Leaf	2-6
XGB	Min Child Weight	1-15
	Learning Rate	0-1
	Subsample	0.5-1
	Colsample by Tree	0.5-1

Table 4.8: Range of Hyperparameters for Each Model

4.1.5.3 Predictive Models

In this thesis, four different models were trained to gain a better understanding of the effect of each set of features. These included the following models:

1. Model 1: Used only static features and gave an estimate of the frequency prior to the start of the placement.
2. Model 2: Used dynamic features without timing features.
3. Model 2: Used features from Models 1 and 2.
4. Model 4: Used all the features.

4.1.6 Testing

In this step, all of the models were tested on the test set. The models had not seen these benchmarks during training; therefore these benchmarks could help find the best model and the generalization error. This model could be used later in the deployment stage. The metrics that were used to test the models were:

- MSE
- RMSE
- MAE
- MAPE
- Accuracy

To better understand the predictions of each model, the test set was used to find different evaluation metrics for each group within the set. As mentioned in Section 2.4, the dataset consisted of 12 benchmarks and their variants. The original benchmark and its variants were grouped together.

4.2 Deployment

The best ML model from the previous section was integrated into GPlace. The model was able to calculate F_{max} at each stage during the placement.

4.3 Case Studies

As discussed earlier, STA is widely used as the delay model for timing driven placement. The main bottleneck in STA is that it often has a low degree of accuracy when predicting F_{max}. This can result in a placement that is unable to satisfy the timing constraints after routing. Figure 5.13 shows the difference between the estimations made by STA and the actual post routing frequency. Estimations made by STA have an average and a maximum error of 12% and 30% respectively. These optimistic predictions of timing during placement can be a serious issue after routing if they fail to meet the required timing constraints. In this case, the designer should either repeat the time-consuming steps of place and route using different parameters to control the optimization or, in the worst case scenario, go back to the first step and modify the design. To overcome this problem, the ML model from this thesis was used as a post-routing probe for F_{max} during placement. This was done to give feedback to the placer and guide the optimization strategy. As discussed in Section 2.2, during global placement several iterations of the placement and legalization were performed for timing and routability optimization. A detailed placement of the ISM was then performed to optimize the timing, wirelength and external pin count. Figure 4.7 shows how the ML model is integrated into the GPlace flow. After each iteration of the global and the detailed placement, the F_{max} of the

placement was calculated as a probe. This was then used in the next iteration to guide the placer. The predictions from the model can be used as feedback to the circuit to guide the placer. There are three different approaches that the placer can use for these predictions:

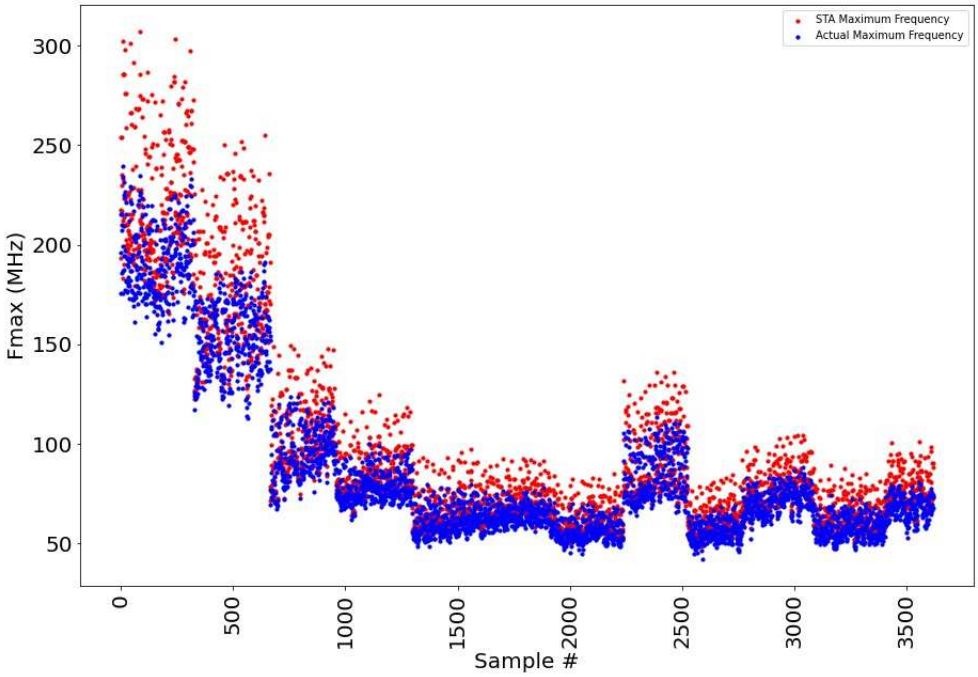


Figure 4.6: Comparison Between Post-routing and STA Estimation of Maximum Frequency

1. The timing driven flow aims to reach a target frequency by reducing the most critical paths in the placement based on the STA. However, experiments have shown that there is a significant gap between the STA and the post routing frequency. As mentioned in Section 2.2.2, the timing driven placer calculates the criticality of

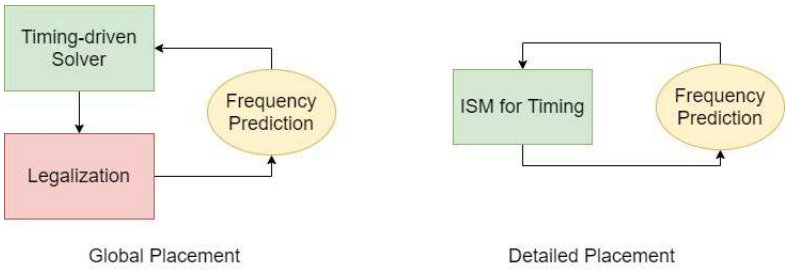


Figure 4.7: Integrating the ML model into GPlace Flow

each path and then uses this criticality to compute the timing costs. The author of this thesis worked to artificially increase the criticality of critical paths having more than a 0.9 criticality proportional to the gap between the STA and the predicted frequency. By considering the most critical paths, it was ensured that most of the paths that might have been the critical path after routing were captured. Then, by increasing the criticality of these paths, the placer was guided to do more timing optimizations in these areas. Criticality was now computed with:

$$crit(sl,i) = \begin{cases} \alpha crit(sl,i) & \text{if } crit(sl,i) > 0.9 \\ crit(sl,i) & \text{otherwise} \end{cases} \tag{4.1}$$

$$\alpha = \frac{STA\ \ Fmax}{Estimated\ \ Fmax} \tag{4.2}$$

2. The second possible use of the framework is the early termination of the placement process. The frequency is tracked, and when it is noted that the target could be reached, the process is stopped. In this process, however, it might be necessary to use different probes, such as routability prediction, to ensure that the placement is

routable.

3. The third possible use of the framework is to decide whether to perform the routing phase or not. Despite using the first method to increase the F_{max}, the target frequency might not be reached due to the limitations of the placer. In this case, the time-consuming step of routing could be avoided, and the placement process could be restarted using another flow.

In this thesis, the second and the third approaches were not used since the target frequency is highly dependant on the application. This thesis used the first approach and identified the gap between the STA and the post-routing frequency to increase the timing optimization and reach a better frequency.

4.4 Summary

This chapter introduced all the necessary steps, such as data preparation and feature engineering, that are required to train and test an ML framework to predict frequency during placement. The case studies were then introduced. Finally, it was discussed how the framework could be integrated into GPlace4.0 to enhance the placement flow and improve the results.

Chapter 5

Results

This chapter introduces the results of this study. First, in Section 5.1, the experimental setup is introduced. Then, in Section 5.2, the results for all the ML models are presented. Finally, in Section 5.3, the effectiveness of the timing probe in GPlace4.0 is discussed.

5.1 Experimental Setup

The ML models used in this report were developed using the Scikit learn [29] library in Python. The source code of GPlace4.0 was provided by [14]. The code was implemented using the C programming language, and compiled using the GCC (Red Hat 4.4.7-18) compiler. All the experiments were carried out using a Linux machine (CentOS release 6.9) running on an Intel (Xeon CPU E3-1270 V2 @ 3.50GHz) processor. The placement tool was run on an Intel (Xeon CPU E3-1270 v5 @ 3.60GHz) processor with 16 GB of RAM. The placement solutions produced by GPlace4.0 were routed using Xilinx Vivado v2015.4, with a patch applied to make Vivado compatible with the modified Bookshelf

Format used by the GPlace4.0 placement tool.

5.2 ML Framework Results

In this section, the results of the ML models used to predict F_{Max} are presented. As mentioned in Section 4.1.5, this study used four models that were based on the features being employed. These models were:

- Model 1: Uses static features
- Model 2: Uses dynamic features (without timing features)
- Model 3: Uses features from Models 1 and 2
- Model 4: Uses all features

For each of the models, the results of the feature selection and the hyperparameter tuning were first presented. Then the prediction quality of each model was mentioned using the metrics introduced in Section 4.1.6. The MAPE and the MSE were also calculated for each group of FPGAs separately. In the end, the labels and the predicted values were drawn on the same plot to visualize the prediction quality. As well, the loss function of the model vs the size of the training set was plotted to detect possible overfitting.

5.2.1 Model 1

5.2.1.1 Feature Selection

Table 5.1 shows the results of the feature selection for Model 1.

Benchmark Group	LR	SVM	DT	RF	XGB	SC	Used By
LUTS	*		*	*		*	4
LUT2	*			*		*	3
LUT3	*						1
LUT4	*						1
LUT5	*						1
LUT6	*						1
FF	*		*	*	*	*	5
DSP					*	*	2
RAM				*	*	*	3
IO					*		1
CSET			*	*	*	*	4
NET2	*	*	*	*	*	*	6
NET3	*	*	*	*	*	*	6
NET4	*	*			*		3
NET5-10	*	*					2
NET10-30	*	*					2
NET30	*	*	*	*		*	5
NETS	*	*	*	*		*	5
PINS	*	*	*		*	*	5
LP	*	*	*	*	*		5
LPMEAN	*	*	*	*	*	*	6
LPSTD			*	*	*	*	4
LP10		*	*	*			3
LP20	*	*			*	*	4
Total	18	12	12	13	13	14	

Table 5.1: Model 1 Feature Selection Result

5.2.1.2 Hyperparameter Tuning

Table 5.2 shows the results of hyperparameter tuning on Model 1.

Model	Hyperparameter	Best Value
LR	Learning Rate	0.1
SVM	Kernel	Radial Basis Function
	Regularization Param	20
	Kernel Coefficient	scale
DT	Criterion	MSE
	Max Depth	15
	Min Samples Split	4
	Max Features	Sqrt
RF	Criterion	MSE
	Number of Estimators	20
	Min Samples Split	10
	Min Samples Leaf	6
XGB	Min Child Weight	10
	Learning Rate	0.3
	Subsample	1
	Colsample by Tree	1

Table 5.2: Best Hyperparameters for Model 1

5.2.1.3 Model Results

Table 5.3 shows the prediction quality obtained by each of the ML models before performing feature selection and hyperparameter tuning. For the simple models, the LR had the best performance. It could achieve accuracy, MSE, RMSE, MAE and MAPE values of 0.936, 173.05, 13.15, 10.08 and 15.53, respectively. The XGBoost (XGB) had the best performance of all the models and could reach accuracy, MSE, RMSE, MAE and MAPE values of 0.962, 102.37, 10.11, 7.14, and 10.58, respectively.

Model	Accuracy	MSE	RMSE	MAE	MAPE
LR	0.936	173.05	13.15	10.08	15.53
DT	0.935	178.22	13.35	10.54	16.28
SVM	0.905	271.70	16.27	11.31	18.12
RF	0.944	150.33	12.26	8.29	12.96
XGB	**0.962**	**102.37**	**10.11**	**7.14**	**10.58**
SC	0.947	111.93	10.58	7.87	11.23

Table 5.3: Model 1 Initial Results

Table 5.4 shows the prediction quality obtained by each of the ML models after performing feature selection and hyperparameter tuning. All of the models benefited from feature selection and hyperparameter tuning as their performance in all the metrics increased. For the simple models, DT had the most improvement and could reach accuracy, MSE, RMSE, MAE and MAPE values of 0.953, 127.41, 11.28, 7.67, and 12.27, respectively. The XGB had the best performance in all of the groups with accuracy, MSE, RMSE, MAE and MAPE values of 0.964, 96.49, 9.82, 6.62, and 10.19, respectively.

Model	Accuracy	MSE	RMSE	MAE	MAPE
LR	0.936	173.05	13.15	10.08	15.53
DT	0.953	127.41	11.28	7.67	12.27
SVM	0.919	247.53	15.73	10.88	17.47
RF	0.958	112.64	10.61	7.08	11.03
XGB	**0.964**	**96.49**	**9.82**	**6.62**	**10.19**
SC	0.947	102.6	10.13	7.00	10.89

Table 5.4: Model 1 Final Results

Tables 5.5 and 5.6 show the list of MSE and MAPE for all the groups. The DT-based models, such as RF, XGB and DT, performed better than the other models in all of the groups. When MSE is considered, Groups 1, 2, 5, 9 and 12 have the highest MSE. This could be due to several reasons. First, FPGAs 1 and 2 have the highest frequency in the whole dataset. The other groups are mostly low-frequency benchmarks. Therefore, one potential explanation is that the model does not see enough samples of high-frequency circuits. Another potential reason is that the frequency of the benchmarks in Groups 1 and 2 is more scattered.

With regards to the MAPE, even though there is a high MSE in FPGAs 1 and 2, the MAPE is still low, But the results are worse in FPGAs 5, 9 and 12; there is a high MSE and MAPE error in these groups. To give an example, the MAPE can reach up to 40% in FPGA 9. Through prior knowledge, it has been determined that these are congested benchmarks, and congestion plays an important role in determining the F_{max}. Still, there are no dynamic features to capture this effect, and this could be another potential reason for the high degree of error.

Benchmark Group	MSE					
	LR	SVM	DT	RF	XGB	SC
FPGA-1	555.34	21518.18	131.75	297.19	**124.13**	209.91
FPGA-2	599.81	7686.79	**23.36**	179.67	404.37	289.32
FPGA-3	73.75	593.09	56.39	**20.403**	29.28	25.46
FPGA-4	33.47	95.88	**22.47**	31.885	35.37	32.37
FPGA-5	468.42	881.67	76.78	**72.72**	130.135	90.82
FPGA-6	32.48	96.49	5.35	4.03	**3.59**	4.35
FPGA-7	21.94	523.85	22.77	**14.61**	20.04	19.45
FPGA-8	37.143	261.93	**8.31**	20.45	16.47	17.28
FPGA-9	240.833	1762.38	**193.34**	251.84	223.08	204.56
FPGA-10	72.71	17.49	15.86	15.01	**2.00**	7.49
FPGA-11	62.76	368.04	20.71	5.57	**1.07**	8.67
FPGA-12	187.40	609.89	458.63	**295.36**	354.17	372.08

Table 5.5: Model 1 MSE for Each Group

Benchmark Group	MAPE					
	LR	SVM	DT	RF	XGB	SC
FPGA-1	8.50	63.90	5.03	6.74	**4.34**	4.78
FPGA-2	11.86	50.70	**2.82**	6.27	10.77	5.34
FPGA-3	6.21	22.09	6.52	**4.15**	4.73	5.28
FPGA-4	4.95	7.35	**4.30**	5.71	6.26	4.57
FPGA-5	42.24	56.92	15.65	**13.71**	18.16	15.96
FPGA-6	7.31	13.19	2.33	2.39	**2.22**	2.35
FPGA-7	7.25	38.71	**5.61**	5.80	6.37	6.07
FPGA-8	5.71	16.47	2.38	**0.75**	3.80	2.14
FPGA-9	40.54	123.01	**38.97**	43.62	41.91	40.47
FPGA-10	7.84	4.84	3.65	3.48	**1.66**	2.93
FPGA-11	11.82	30.42	4.68	2.69	**1.59**	3.11
FPGA-12	20.36	42.41	35.51	**20.27**	23.74	30.17

Table 5.6: Model 1 MAPE for Each Group

Table 5.7 shows the training and inference time for each model. In simple models, LR has the least training and inference time, while among ensembles, RF has the best performance.

Model	LR	SVM	DT	RF	XGB	SC
Training Time (s)	0.002	0.03	0.007	0.05	0.12	1.71
Inference Time (s)	0.001	0.002	0.001	0.01	0.001	0.05

Table 5.7: Model 1 Training and Inference Time

Figure 5.1 shows the actual and the predicted frequency for each placement. The order was plotted according to the number of the group, and different groups are separated with black lines. In comparison, Figure 5.2 shows the same results in the order of the actual frequency, from lowest to highest.

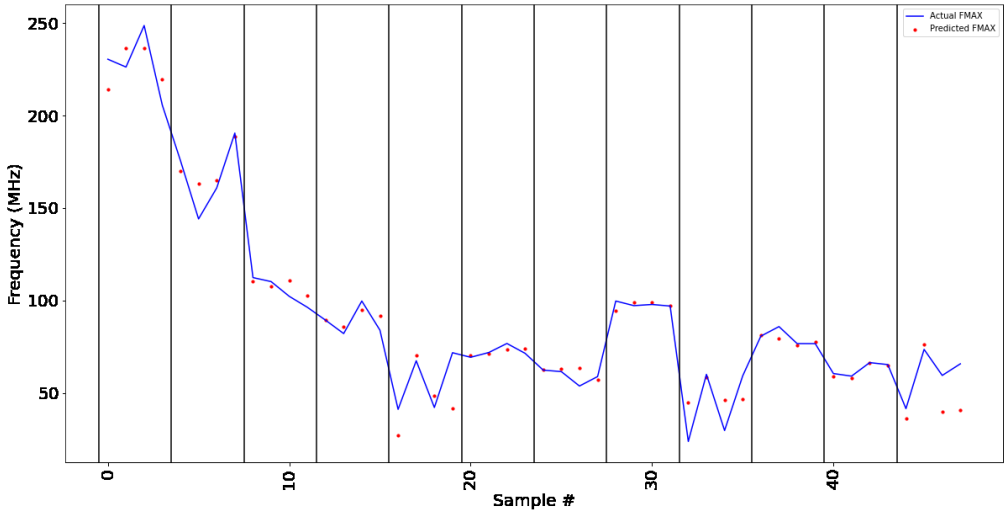


Figure 5.1: Actual and Predicted Frequency for Model 1

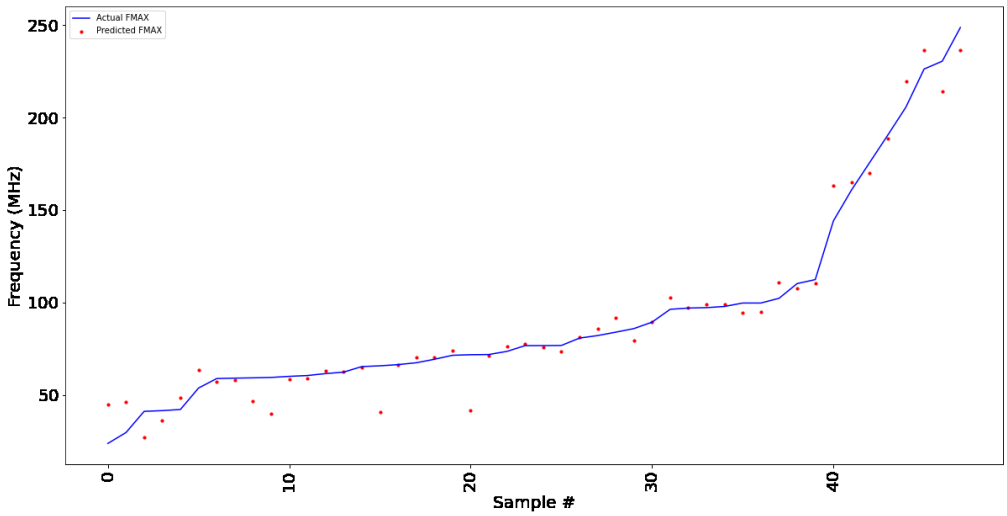


Figure 5.2: Actual and Predicted Frequency for Model 1

Figure 5.3 shows the MSE of each ML model affected by the size of the training set. First, smaller series of training sets were randomly chosen ranging in size from 10 to 300. Each new training set was then used to train the model. Finally, each model was tested using the test set. The models reach within 5% of their final error after 150 samples.

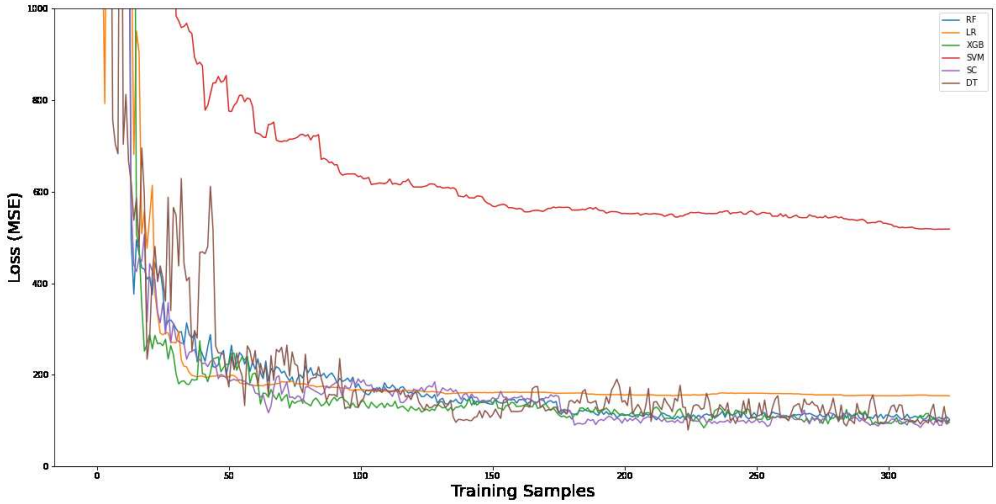


Figure 5.3: Loss Function for Model 1

5.2.2 Model 2

5.2.2.1 Feature Selection

Table 5.8 shows the results of the feature selection for Model 2.

Benchmark Group	LR	SVM	DT	RF	XGB	SC	Used By
HPWL		*	*	*	*	*	5
SW	*	*	*	*	*	*	6
CONGMEAN	*	*	*	*	*	*	6
CONGSTD	*		*	*	*	*	5
WLPASTD		*		*	*	*	4
WLPAMEAN		*	*	*	*	*	5
NCPR5STD	*	*	*	*	*		5
NCPR5MEAN	*	*	*	*	*	*	6
NCPR9STD	*	*	*	*	*		5
NCPR9MEAN	*	*	*	*	*	*	6
PINSTD	*		*		*	*	4
PINMEAN	*	*	*		*	*	5
Total	10	10	11	10	12	10	

Table 5.8: Model 2 Feature Selection Results

5.2.2.2 Hyperparameter Tuning

Table 5.9 shows the results of the hyperparameter tuning on Model 2.

Model	Hyperparameter	Best Value
LR	Learning Rate	0.1
SVM	Kernel	Radial Basis Function
	Regularization Param	15
	Kernel Coefficient	Scale
DT	Criterion	MSE
	Max Depth	12
	Min Samples Split	6
	Max Features	Sqrt
RF	Criterion	MSE
	Number of Estimators	20
	Min Samples Split	7
	Min Samples Leaf	4
XGB	Min Child Weight	10
	Learning Rate	0.3
	Subsample	1
	Colsample by Tree	0.5

Table 5.9: Best Hyperparameters for Model 2

5.2.2.3 Model Results

Table 5.10 shows the prediction quality obtained by each of the ML models before performing the hyperparameter tuning. Among the simple models, DT had the best performance and could reach accuracy, MSE, RMSE, MAE, and MAPE values of 0.972, 51.56,

7.18, 4.74, and 5.42, respectively. The performance of the XGB and the RF was close. Overall, RF had the best performance and could reach accuracy, MSE, RMSE, MAE, and MAPE values of 0.980, 36.20, 6.01, 5.66, and 4.55, respectively.

Model	Accuracy	MSE	RMSE	MAE	MAPE
LR	0.946	98.78	9.93	7.33	8.82
SVM	0.911	162.60	12.75	9.23	10.85
DT	0.972	51.56	7.18	4.74	5.42
RF	**0.980**	**36.20**	**6.01**	**5.66**	**4.55**
XGB	0.979	38.5	6.21	5.6	4.74
SC	0.980	46.10	6.79	5.62	4.86

Table 5.10: Model 2 Initial Results

Table 5.11 shows the prediction quality obtained by each of the ML models after performing hyperparameter tuning. All the models benefited from hyperparameter tuning, and their performance in all the metrics was increased. XGB now has the best performance, reaching accuracy, MSE, RMSE, MAE and MAPE values of 0.981, 33.75, 5.82, 4.26, and 4.56, respectively.

Model	Accuracy	MSE	RMSE	MAE	MAPE
LR	0.946	98.78	9.93	7.33	8.82
SVM	0.923	140.37	11.84	8.64	10.39
DT	0.974	47.56	6.89	4.52	5.11
RF	0.981	34.54	5.87	4.06	4.44
XGB	**0.981**	**33.75**	**5.82**	**4.26**	**4.56**
SC	0.980	33.98	5.83	4.55	4.58

Table 5.11: Model 2 Final Results

Tables 5.12 and 5.13 show the MSE and the MAPE for each group.

For the MSE, the error in Groups 1, 2, 5, 9 and 12 has been improved by using the dynamic features. The most significant decrease was in the congested groups, which includes 5, 9, and 12. Now the highest error is found in Groups 1, 2, 3, and 8, which have values of 98.62, 87.55, 43.01, and 92.13, respectively. The main reason for this is that these are less congested placements, and the congestion features do not give any new information to the model. As well, there is a small switch utilization, and different features, including WLPA and NCPR, are zero for most switches.

The most significant decrease was in the MAPE of Groups 5, 9, and 12; for example, the MAPE was reduced from 43% to 4% in FPGA 5. This improvement is important because the purpose of using dynamic features was to decrease errors in the congested placements. The highest error was for FPGA 8, which had a value of 9.26. An attempt will be made later to further decrease the errors in these benchmarks by adding other features.

Benchmark Group	MSE					
	LR	SVM	DT	RF	XGB	SC
FPGA-1	305.92	224.78	148.52	106.03	**98.62**	109.76
FPGA-2	315.82	279.03	118.95	97.17	**87.55**	101.27
FPGA-3	68.56	119.35	63.40	44.45	**42.01**	51.56
FPGA-4	42.50	78.95	26.89	16.05	**14.00**	20.14
FPGA-5	36.16	41.62	15.12	10.53	**10.47**	11.54
FPGA-6	110.95	152.96	13.33	7.52	**7.41**	9.26
FPGA-7	127.19	58.43	16.54	15.32	**16.14**	16.48
FPGA-8	269.27	642.12	140.25	104.52	**92.13**	95.65
FPGA-9	38.38	25.86	12.68	**4.29**	4.34	6.79
FPGA-10	34.13	23.71	11.77	**10.55**	12.91	14.46
FPGA-11	26.67	23.72	14.25	**7.96**	8.25	10.57
FPGA-12	55.22	26.17	16.51	**10.18**	11.27	15.21

Table 5.12: Model 2 MSE for Each Group

Benchmark Group	MAPE					
	LR	SVM	DT	RF	XGB	SC
FPGA-1	6.68	5.85	4.56	**4.04**	4.06	4.43
FPGA-2	9.85	9.23	5.44	5.31	**5.11**	5.72
FPGA-3	7.44	8.58	6.17	**5.21**	5.25	6.12
FPGA-4	7.06	8.75	**4.63**	3.92	3.50	4.28
FPGA-5	8.71	7.79	5.12	**3.99**	4.11	4.02
FPGA-6	15.17	19.03	4.53	**3.49**	3.58	3.69
FPGA-7	19.54	12.57	**4.70**	5.36	5.70	5.27
FPGA-8	15.51	25.58	11.56	9.66	**9.26**	10.22
FPGA-9	10.37	7.93	4.58	2.88	**2.85**	3.57
FPGA-10	6.87	5.49	3.65	**3.62**	3.79	3.89
FPGA-11	7.33	7.44	4.76	**3.77**	3.88	4.27
FPGA-12	8.92	6.41	4.36	**3.64**	3.97	3.78

Table 5.13: Model 2 MAPE for Each Group

Table 5.14 shows the training and the inference time for each model. Among the simple models, LR had the lowest training and inference time, while among the ensembles, RF had the best performance.

Model	LR	SVM	DT	RF	XGB	SC
Training Time (s)	0.006	1.23	0.07	0.12	0.34	1.94
Inference Time (s)	0.001	0.45	0.001	0.02	0.005	0.07

Table 5.14: Model 2 Training and Inference Time

Figure 5.4 shows the actual and the predicted frequency for each placement. The order of the predicted frequency was plotted according to the number of the group, and is divided by black lines. In comparison, Figure 5.5 shows the same results in the order of the actual frequency, ranging from lowest to highest.

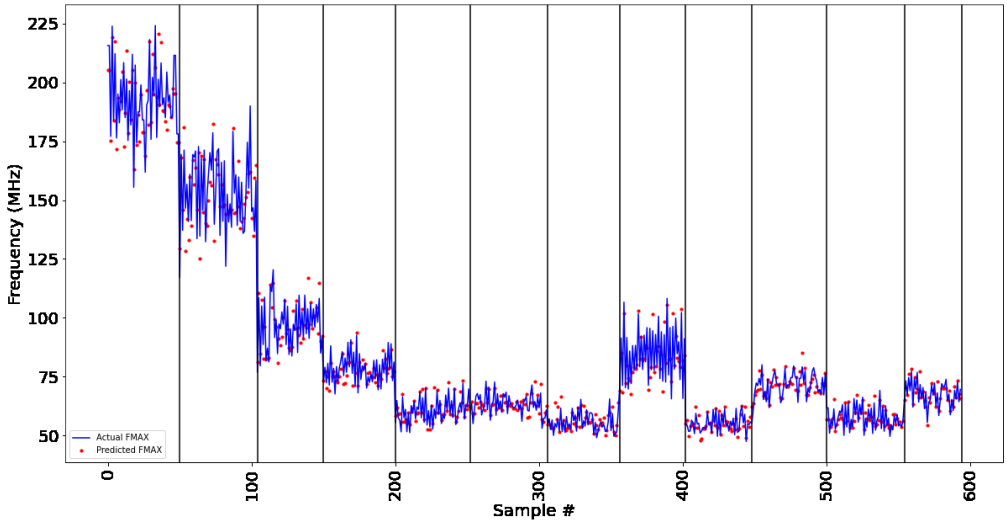


Figure 5.4: Actual and Predicted Frequency for Model 2

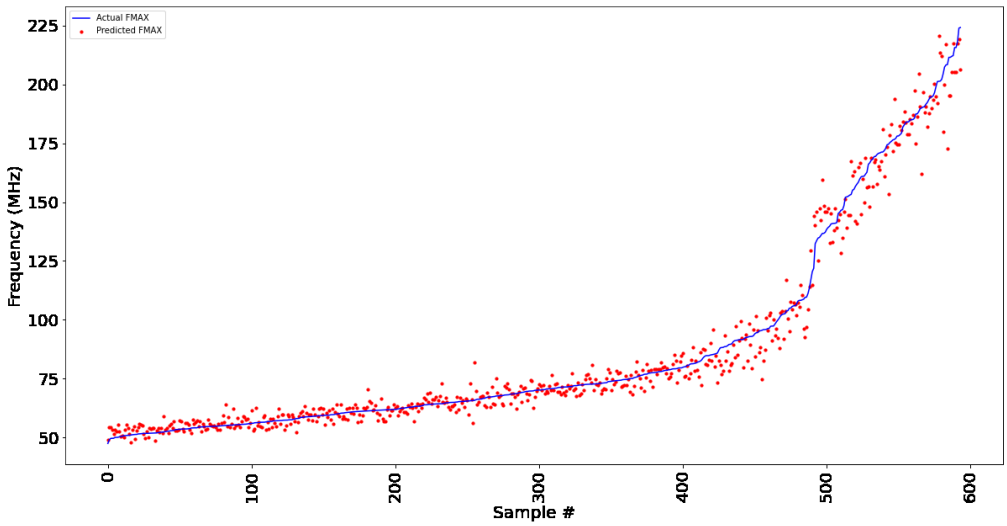


Figure 5.5: Actual and Predicted Frequency for Model 2

Figure 5.6 shows the MSE of each ML model affected by the size of the training set. First, smaller series of training sets were randomly chosen between the size of 10 to 3000. Each new training set was then used to train the model. Finally, each model was tested using the test set. The models arrived within 5% of their final error after 2000 samples.

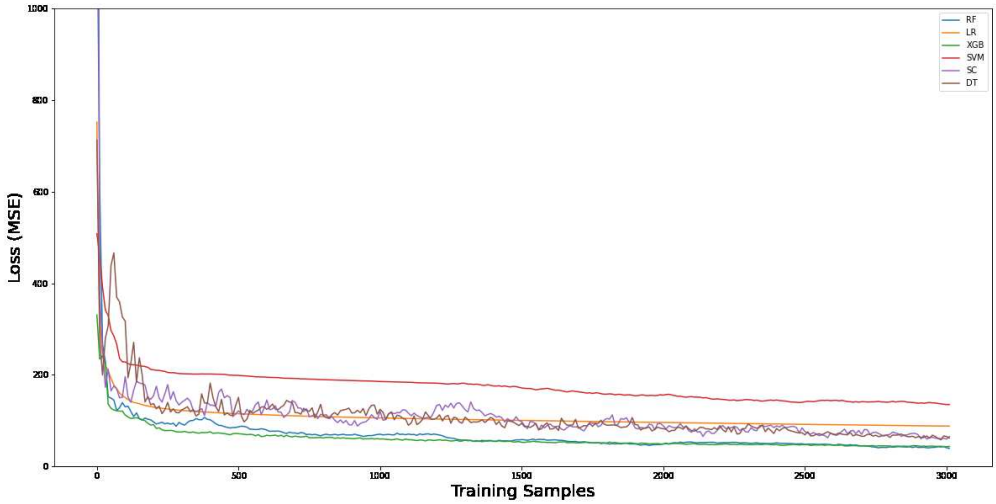


Figure 5.6: Loss Function for Model 2

5.2.3 Model 3

5.2.3.1 Feature Selection

Table 5.15 shows the results of the feature selection on Model 3.

Benchmark Group	LR	SVM	DT	RF	XGB	SC	Used By
LUTS	*		*		*	*	4
LUT2	*	*	*	*		*	5
LUT3	*		*	*		*	4
LUT4	*	*	*			*	4
LUT5	*		*			*	3
LUT6	*		*	*			3
FF			*	*	*	*	4
DSP		*	*	*	*	*	5
RAM	*	*	*		*		4
IO	*	*	*				3
CSET	*	*		*	*	*	4
NET2			*	*	*	*	4
NET3			*	*	*	*	4
NET4	*	*	*	*	*		5
NET5-10	*	*	*				3
NET10-30	*	*	*	*	*	*	6
NET30	*	*	*	*	*	*	6
NETS	*	*			*		3
PINS	*	*			*	*	4
LP		*	*		*		3
LPMEAN	*	*		*	*	*	5
LPSTD		*		*		*	3
LP10	*	*		*			3
LP20	*	*		*		*	4
HPWL	*	*	*		*	*	5
SW	*	*	*	*	*	*	6
CONGMEAN	*	*	*		*		4
CONGSTD		*	*			*	3
WLPASTD	*	*		*	*		4
WLPAMEAN	*	*		*	*		4
NCPR5STD	*	*				*	3
NCPR5MEAN	*	*	*	*	*		5
NCPR9STD		*	*	*	*	*	5
PINSTD	*	*		*	*	*	5
PINMEAN	*	*	*		*	*	5
Total	28	28	24	21	24	23	

Table 5.15: Model 3 Feature Selection Results

5.2.3.2 Hyperparameter Tuning

Table 5.16 shows the results of hyperparameter tuning on Model 3.

Model	Hyperparameter	Best Value
LR	Learning Rate	0.1
SVM	Kernel	Radial Basis Function
	Regularization Param	15
	Kernel Coefficient	Scale
DT	Criterion	MSE
	Max Depth	20
	Min Samples Split	6
	Max Features	Sqrt
RF	Criterion	MSE
	Number of Estimators	20
	Min Samples Split	10
	Min Samples Leaf	4
XGB	Min Child Weight	10
	Learning Rate	0.3
	Subsample	0.6
	Colsample by Tree	0.75

Table 5.16: Best Hyperparameters for Model 3

5.2.3.3 Model Results

Table 5.17 shows the prediction quality obtained by each of the ML models before performing hyperparameter tuning. Among simple models, DT has the best performance and could reach accuracy, MSE, RMSE, MAE, and MAPE values of 0.979, 38.14, 6.17, 4.03, and 4.57, respectively. The XGB and RF have a similar performance; however, RF performs better overall, and could reach accuracy, MSE, RMSE, MAE, and MAPE values of 0.985, 27.54, 5.24, 3.48, and 3.80, respectively.

Model	Accuracy	MSE	RMSE	MAE	MAPE
LR	0.969	56.86	7.53	5.47	6.28
SVM	0.926	135.24	11.62	8.21	9.42
DT	0.979	38.14	6.17	4.03	4.57
RF	**0.985**	**27.54**	**5.24**	**3.48**	**3.80**
XGB	0.983	30.39	5.51	3.74	4.12
SC	0.980	36.12	6.01	3.95	4.41

Table 5.17: Model 3 Initial Results

Table 5.18 shows the prediction quality obtained by each of the ML models after performing hyperparameter tuning and feature selection. All the models benefited from hyperparameter tuning, and their performance in all the metrics was increased. The XGB now had the best performance, and could reach accuracy, MSE, RMSE, MAE and MAPE values of 0.986, 25.77, 5.07, 3.28, and 3.57, respectively.

Model	Accuracy	MSE	RMSE	MAE	MAPE
LR	0.969	56.86	7.53	5.47	6.28
SVM	0.958	76.13	8.72	6.18	7.02
DT	0.980	36.64	6.05	4.00	4.55
RF	0.985	26.97	5.19	3.43	3.75
XGB	**0.986**	**25.77**	**5.07**	**3.28**	**3.57**
SC	0.981	28.83	5.37	3.86	4.12

Table 5.18: Model 3 Final Results

As expected, Model 3 performed better than both of the previous models, and the XGB and RF had the best performance in all of the groups. To better understand the performance of this model, the error within each group was investigated. The MSE and the MAPE of all models is listed in Tables 5.20 and 5.19. The percentage error in the congested groups has been improved, and there was a lower squared error in Groups 1, 2, and 3. This means that the benefits of both models could be captured.

The MAPE error in the XGB was less than 7% for all of the groups. The highest MSE was for Groups 1, 2, 3, and 8, which was 80.01, 78.13, 42.13, and 50.77, respectively. Although this error was reduced from Model 2, a further attempt to reduce the error in these groups will be made by adding new features.

Benchmark Group	MAPE					
	LR	SVM	DT	RF	XGB	SC
FPGA-1	5.21	5.83	3.99	3.40	**3.30**	3.78
FPGA-2	8.10	8.71	5.04	5.05	**4.67**	4.92
FPGA-3	6.51	8.00	5.52	**4.75**	4.78	5.13
FPGA-4	5.33	5.95	5.27	4.05	**3.85**	4.98
FPGA-5	6.01	6.48	4.43	**3.26**	3.32	3.88
FPGA-6	4.16	5.62	4.63	3.53	**3.15**	3.42
FPGA-7	6.85	5.81	2.80	2.65	**2.50**	2.70
FPGA-8	9.77	11.84	8.44	7.52	**6.57**	7.15
FPGA-9	6.00	8.04	4.66	**2.67**	2.68	3.12
FPGA-10	6.64	5.98	3.03	**2.58**	2.59	2.94
FPGA-11	5.43	6.78	4.28	3.12	**2.86**	3.63
FPGA-12	5.54	5.49	2.85	**2.51**	2.81	2.85

Table 5.19: Model 3 MAPE for Each Group

Benchmark Group	MSE					
	LR	SVM	DT	RF	XGB	SC
FPGA-1	161.89	215.99	104.44	84.14	**80.01**	90.43
FPGA-2	206.02	245.12	98.59	82.83	**78.13**	80.45
FPGA-3	59.64	97.10	57.45	**39.91**	42.13	40.69
FPGA-4	24.89	35.08	34.58	**17.14**	18.67	19.76
FPGA-5	18.98	22.01	12.79	**6.49**	7.13	8.95
FPGA-6	11.32	18.22	14.54	7.37	**6.75**	6.76
FPGA-7	23.45	15.79	5.26	3.66	**3.54**	4.21
FPGA-8	84.23	164.38	88.09	61.60	**50.77**	60.76
FPGA-9	15.08	26.22	16.45	3.45	**3.45**	3.56
FPGA-10	30.16	26.34	7.87	**5.70**	6.13	5.97
FPGA-11	13.78	20.59	10.19	4.92	**4.83**	4.90
FPGA-12	21.96	19.95	6.41	**4.47**	5.74	5.32

Table 5.20: Model 3 MSE for Each Group

Table 5.21 shows the training and the inference time for each model. Among simple models, LR required the least training and inference time. In comparison, among the ensembles, RF had the best performance.

Model	LR	SVM	DT	RF	XGB	SC
Training Time (s)	0.007	1.26	0.07	0.19	0.63	2.17
Inference Time (s)	0.001	0.47	0.001	0.007	0.005	0.08

Table 5.21: Model 3 Training and Inference Time

Figure 5.7 shows the actual and the predicted frequency for each placement. The predicted frequency was plotted using the order of the group's number and is divided by black lines. In comparison, Figure 5.8 shows the same results in the order of the actual frequency, ranging from lowest to highest.

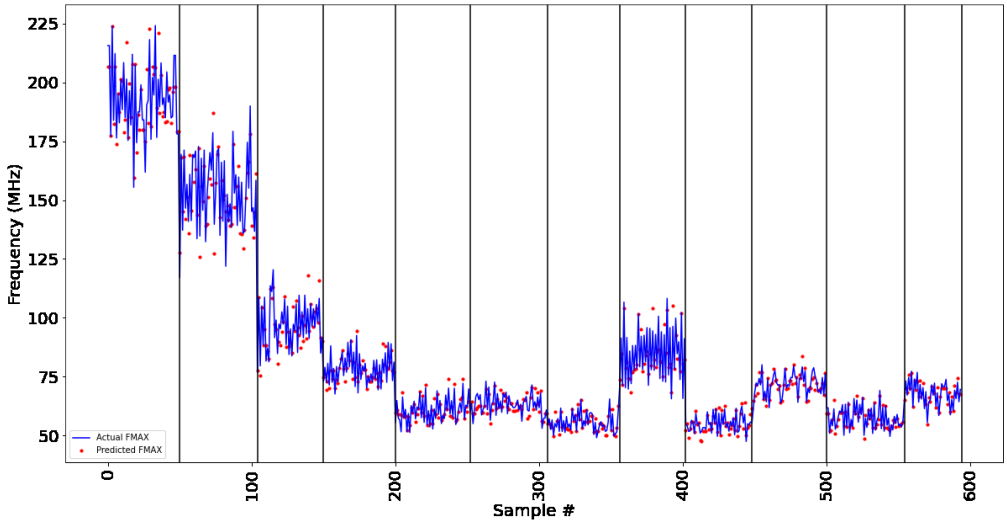


Figure 5.7: Actual and Predicted Frequency for Model 3

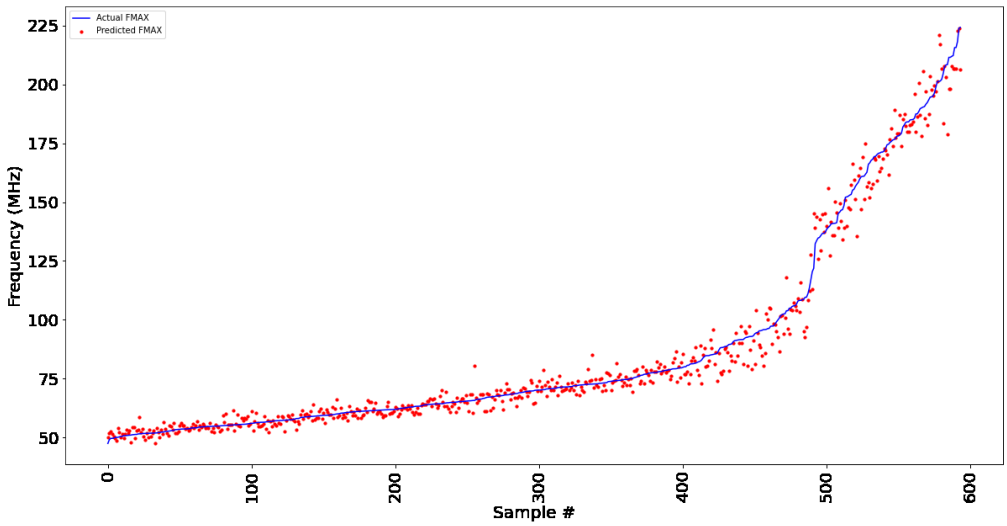


Figure 5.8: Actual and Predicted Frequency for Model 3

Figure 5.9 shows the loss function for all six ML models. They were within 5% of their final error after 1500 samples.

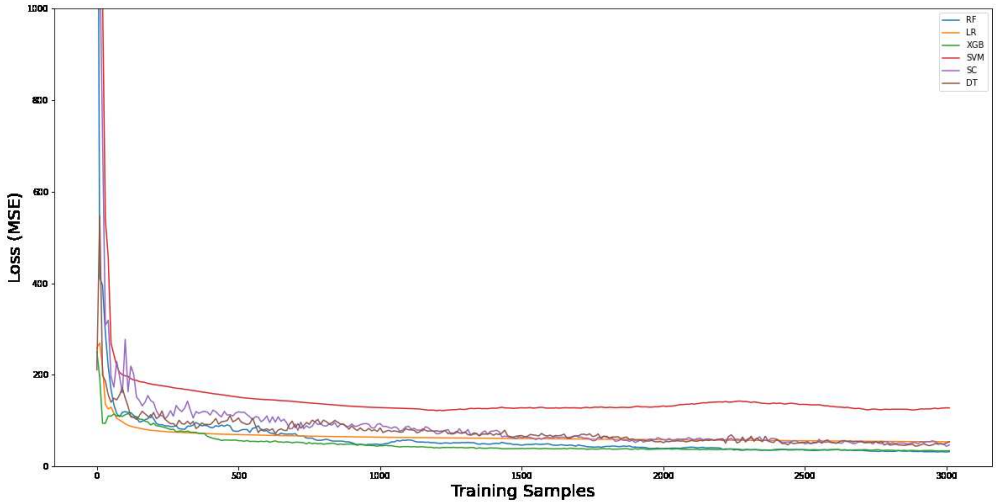


Figure 5.9: Loss Function for Model 3

5.2.4 Model 4

5.2.4.1 Feature Selection

Table 5.22 shows the results of the feature selection on Model 4.

Benchmark Group	LR	SVM	DT	RF	XGB	SC	Used By
LUTS	*		*		*	*	4
LUT2	*	*	*			*	3
LUT3	*		*			*	3
LUT4	*	*	*		*	*	4
LUT5	*		*		*	*	3
LUT6	*		*	*	*		3
FF	*		*	*	*	*	5
DSP			*	*	*	*	4
RAM		*	*		*		3
IO		*	*				3
CSET		*		*	*	*	4
NET2	*		*	*	*	*	4
NET3	*		*	*	*		4
NET4	*	*	*	*			4
NET5-10	*	*	*				3
NET10-30	*	*	*	*		*	5
NET30	*	*	*	*		*	5
NETS		*			*		2
PINS		*				*	2
LP		*	*				2
LPMEAN	*	*		*	*	*	5
LPSTD		*		*		*	3
LP10		*		*	*		3
LP20	*	*			*	*	4
HPWL	*	*	*		*	*	5
SW	*	*	*	*	*	*	6
CONGMEAN	*	*	*		*		4
CONGSTD		*	*			*	3
WLPASTD		*		*	*		3
WLPAMEAN		*		*	*		3
NCPR5STD	*	*				*	3
NCPR5MEAN	*	*	*	*	*		5
NCPR9STD	*	*		*			3
PINSTD	*	*		*	*		4
PINMEAN		*	*	*	*	*	5
CPL	*		*				2
CPD	*	*	*	*	*	*	6
NETMEAN	*			*			2
NETSTD	*		*				2
CPCONGMEAN		*		*	*	*	4
CPCONGSTD	*		*	*	*		3
Total	28	30	24	24	25	23	

Table 5.22: Model 4 Feature Selection Results

5.2.4.2 Hyperparameter Tuning

Table 5.23 shows the results of hyperparameter tuning on Model 4.

Model	Hyperparameter	Best Value
LR	Learning Rate	0.1
SVM	Kernel	Radial Basis Function
	Regularization Param	15
	Kernel Coefficient	Scale
DT	Criterion	MSE
	Max Depth	20
	Min Samples Split	6
	Max Features	Sqrt
RF	Criterion	MSE
	Number of Estimators	20
	Min Samples Split	10
	Min Samples Leaf	4
XGB	Min Child Weight	10
	Learning Rate	0.3
	Subsample	0.6
	Colsample by Tree	0.75

Table 5.23: Best Hyperparameters for Model 4

5.2.4.3 Model Results

Table 5.24 shows the prediction quality obtained by each of the ML models before performing hyperparameter tuning and feature selection. Among the simple models, DT had the best performance and could reach accuracy, MSE, RMSE, MAE, and MAPE values of 0.993, 12.83, 3.58, 2.28, and 2.45, respectively. The performance of the XGB and RF was similar, although RF had the best performance overall. The RF could reach accuracy, MSE, RMSE, MAE, and MAPE values of 0.996, 8.01, 2.83, 1.79, and 1.90, respectively.

Model	Accuracy	MSE	RMSE	MAE	MAPE
LR	0.982	32.38	5.69	3.94	4.58
SVM	0.922	142.36	11.93	8.34	9.55
DT	0.993	12.83	3.58	2.28	2.45
RF	**0.996**	**8.01**	**2.83**	**1.79**	**1.90**
XGB	0.996	8.15	2.85	1.81	1.94
SC	0.995	8.44	2.90	1.86	1.96

Table 5.24: Model 4 Initial Results

Table 5.25 shows the prediction quality obtained by each of the ML models after performing hyperparameter tuning and feature selection. All of the models benefited from hyperparameter tuning, and their performance in all of the metrics was increased. The XGB had the best performance, reaching accuracy, MSE, RMSE, MAE and MAPE values of 0.996, 6.67, 2.58, 1.72, and 1.72, respectively.

Model	Accuracy	MSE	RMSE	MAE	MAPE
LR	0.982	32.38	5.69	3.94	4.58
SVM	0.949	93.83	9.68	6.79	7.56
DT	0.993	12.67	3.56	2.23	2.38
RF	0.996	7.56	2.75	1.73	1.85
XGB	**0.996**	**6.67**	**2.58**	**1.72**	**1.72**
SC	0.995	8.42	2.90	1.85	1.96

Table 5.25: Model 4 Final Results

To better understand the performance in all of the groups, the MSE and the MAPE of all of the models is listed in Tables 5.26 and 5.27. The percentage error for all of the groups was improved by adding the critical path features. In all of the groups, the highest MAPE error for XGB model was 2.71. The MSE was also improved, and the highest error was for Groups 1 and 2 with errors of 24.28 and 28.24, respectively. The higher degree of error in these groups was expected since the frequency was more scattered in them and the prediction was more challenging.

Benchmark Group	MSE					
	LR	SVM	DT	RF	XGB	SC
FPGA-1	118.06	216.30	54.49	29.85	**24.28**	31.07
FPGA-2	125.99	245.55	51.29	31.83	**28.24**	33.25
FPGA-3	14.69	97.04	10.12	11.80	**7.99**	13.17
FPGA-4	10.50	35.05	5.10	2.95	**2.64**	3.42
FPGA-5	12.42	22.01	3.10	2.44	**1.99**	2.44
FPGA-6	12.76	18.15	3.09	**1.72**	1.86	1.89
FPGA-7	16.68	15.75	3.54	1.77	**1.30**	1.82
FPGA-8	11.24	164.70	5.77	4.20	**3.44**	4.31
FPGA-9	22.64	26.24	2.96	1.69	**1.08**	1.48
FPGA-10	7.87	26.34	3.28	2.38	**2.04**	2.42
FPGA-11	15.92	20.57	1.89	1.28	**1.01**	1.23
FPGA-12	8.29	19.94	3.27	**2.12**	2.15	2.37

Table 5.26: Model 4 MSE for Each Group

Benchmark Group	MAPE					
	LR	SVM	DT	RF	XGB	SC
FPGA-1	4.47	5.83	2.84	2.27	**1.97**	2.38
FPGA-2	6.24	8.71	3.90	3.00	**2.71**	3.06
FPGA-3	3.05	7.99	2.54	2.43	**2.11**	2.50
FPGA-4	3.39	5.94	2.45	1.76	**1.78**	1.85
FPGA-5	4.85	6.48	2.10	1.88	**1.72**	1.96
FPGA-6	4.50	5.60	2.17	1.64	**1.57**	1.74
FPGA-7	5.75	5.80	2.57	1.69	**1.50**	1.72
FPGA-8	2.88	11.84	1.89	1.53	**1.44**	1.60
FPGA-9	7.11	8.04	2.10	1.71	**1.38**	1.67
FPGA-10	3.19	5.98	1.84	1.59	**1.49**	1.68
FPGA-11	5.67	6.78	1.89	1.59	**1.34**	1.57
FPGA-12	3.40	5.49	2.11	**1.57**	1.63	1.68

Table 5.27: Model 4 MAPE for Each Group

Table 5.28 shows the training and the inference times for each model. Among the simple models, LR had the lowest training and inference times, while among the ensembles, RF had the best performance.

Model	LR	SVM	DT	RF	XGB	SC
Training Time (s)	0.01	1.41	0.08	0.22	0.85	2.21
Inference Time (s)	0.001	0.16	0.001	0.003	0.006	0.08

Table 5.28: Model 4 Training and Inference Time

Figure 5.10 shows the actual and the predicted frequency for each placement. The predicted frequency is plotted in order of the group's number and is divided by black lines. In comparison, Figure 5.11 shows the same results placed in order of the actual frequency, ranging from the lowest to the highest values.

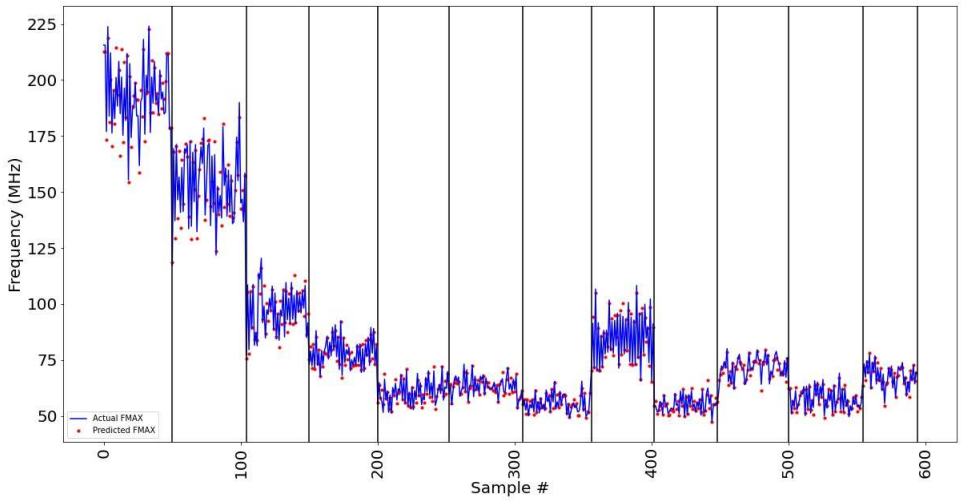


Figure 5.10: Actual and Predicted Frequency for Model 4

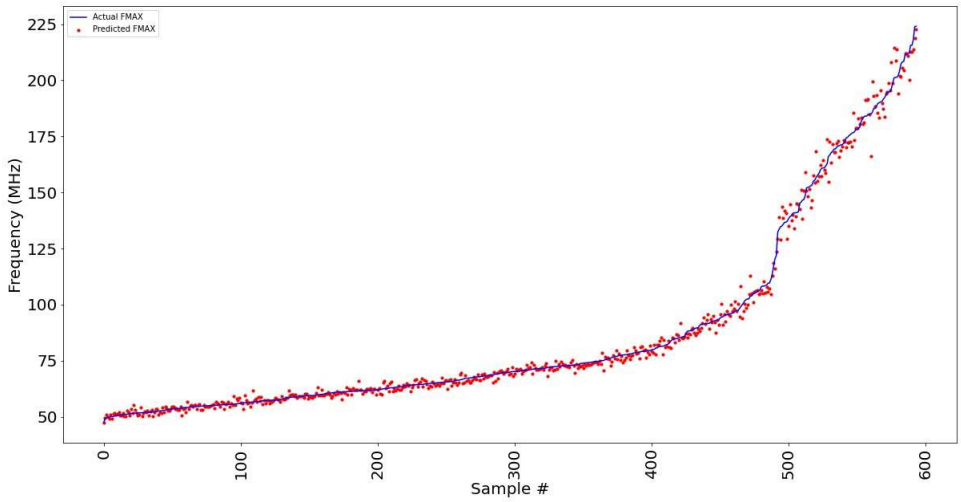


Figure 5.11: Actual and Predicted Frequency for Model 4

Figure 5.12 shows the loss function of all six ML models. They were within 5% of their final error after 1200 samples.

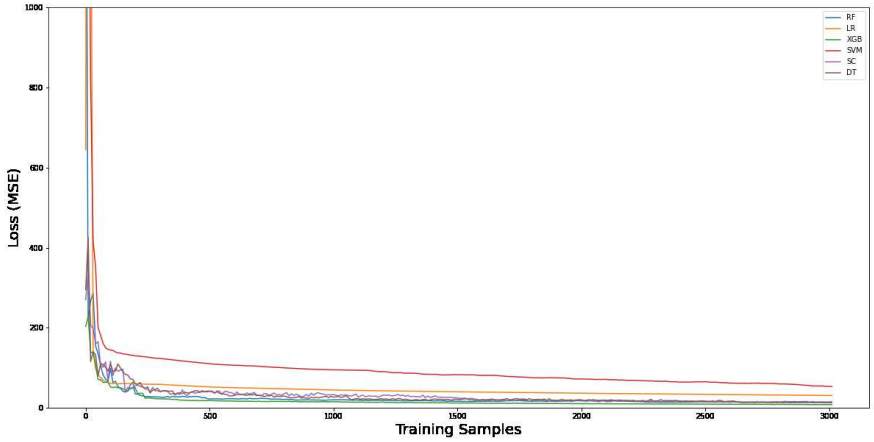


Figure 5.12: Loss Function for Model 4

5.2.5 Analysis

Table 5.29 shows the average, maximum and minimum amount of improvement in all the metrics after performing feature selection and hyperparameter tuning. The table shows that feature selection and hyperparameter tuning are among the most critical steps in the proposed framework. Feature selection has a more significant effect on reducing the prediction error on average, but the amount of improvement that each technique provide us is highly dependant on the ML model we are using. Some models like gradient boosting need a lot of effort on the hyperparameters to avoid overfitting, while simple models like linear regression can benefit most from the feature selection.

Improvement(%)	Accuracy	MSE	RMSE	MAE	MAPE
Mean (FS)	0.35	8.87	4.32	5.83	4.11
Max (FS)	3.21	25.64	16.51	25.37	20.19
Min (FS)	0.15	2.64	1.51	2.22	2.01
Mean (HT)	0.24	5.2	2.62	5.11	3.53
Max (HT)	2.94	24.91	15.63	23.17	19.45
Min (HT)	0.12	2.32	1.64	1.98	1.89

Table 5.29: Improvement after Performing Feature Selection (FS) and Hyperparameter Tuning (HT)

Model 1 used only static features, which were able to produce accurate predictions in most groups. However, the dataset for Model 1 was quite small, and the results might not indicate the exact performance of the model. As well, the error in the congested benchmarks was high: up to an error of 40% in some cases. In comparison, Model 2 only used dynamic features. The accuracy provided by the features for this model were good

in the congested benchmarks. In smaller benchmarks, however, this error was high due to the lack of information. Model 3 used both static and dynamic features. This model benefited from using both features, as the accuracy was increased in all the groups. The highest error was found in Groups 1, 2, 3, and 8. Finally, in Model 4, timing features were added. These features were not accurate post routing critical path, but they were able to reduce the error in Groups 1, 2, 3, and 8 furthermore.

It was noted that after adding a new set of features to a model, the model took fewer samples to arrive within 5% of its final error. Models 2 to 4 reached this level in 2000, 1700, and 1200 samples, respectively. This indicates that each set of new features helped the model to converge faster, and that the features being used were highly correlated with the label.

In terms of the training and the inference times among the ensemble models, RF had the best performance. Among the simple models, LR and DT had a better performance; however, they still did not have a high accuracy when compared to the ensembles. Among all the models that were used, SVM had the lowest accuracy. The main reason for this is that SVM works best when there is a boundary between the different classes. In the dataset used here, however, two benchmarks that are close together in terms of static features may have completely different frequencies, and there is no clear boundary between the groups. LR performed better compared to SVM. However, the DT-based techniques, such as RF, DT, and XGB, outperformed all of the other models. The main reason is that DT-based models are not sensitive to missing data. This could help because there was no dataset available that contained all the possible values for the features. The DT-based models could therefore fill the gaps.

5.3 Case Studies

This section contains a discussion of the effect of integrating the ML framework into GPlace. Section 4.3 contains a discussion of three different applications of the framework during the placement process. However, in this thesis, the second and third approaches were not used since the target frequency is highly dependant on the application. Only the first approach was utilized in order to identify the gap between the STA and the post-routing frequency, and to increase the timing optimization to reach a better frequency. Figure 5.13 shows a comparison between the frequency of the STA and the actual frequency. The MAPE is 12%, which is relatively high in comparison to the model used in this thesis, which has a MAPE of 1.72%.

The framework was used in the placement engine to increase the criticality of the nets proportional to the gap between the STA and the actual frequency. The framework was also used to guide the placer to increase its optimizations. Table 5.30 shows a comparison between the traditional flow of GPlace4.0 and the new flow, which uses ML. The frequency for most benchmarks was increased and a mean improvement of 3% was shown through all the benchmarks. The frequency for all of the benchmarks was improved by using the proposed methodology; however, this improvement was different for each benchmark. The main idea behind this methodology was to increase the criticality of the paths with more than a 0.9 criticality. The goal was to obtain the post routing critical path in this range. In some cases, the post routing critical path was not in this range, or other paths became critical later, so the improvement was less. One solution could involve increasing this range, but experiments have shown that this would lead to an increased focus on timing optimizations, resulting in a placement that has more

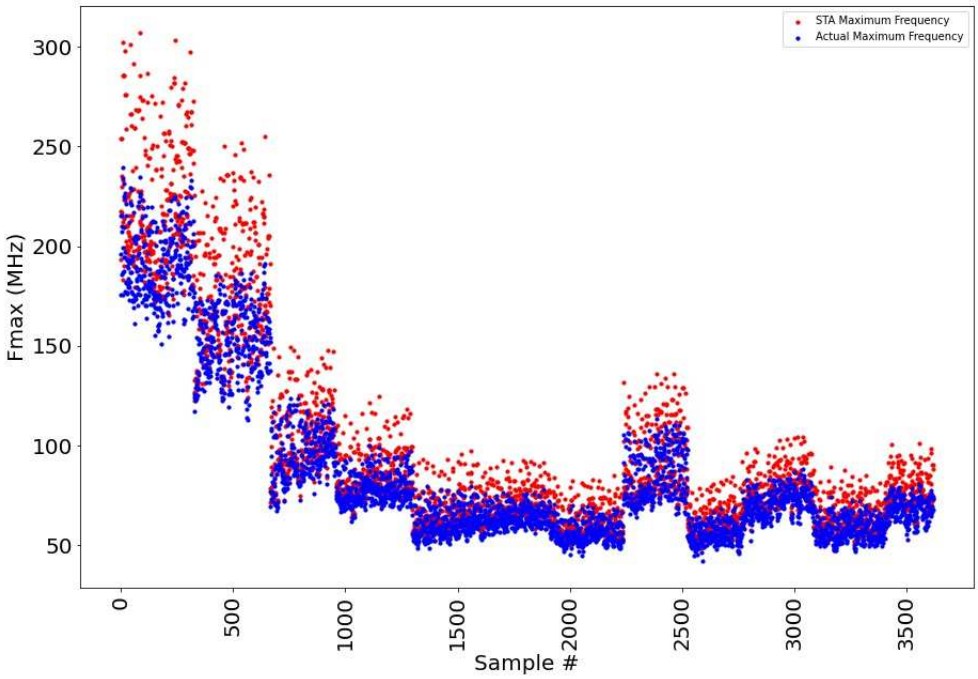


Figure 5.13: Comparison Between Post-routing and STA Estimation of Maximum Frequency

wirelength and is difficult to route.

Benchmark Group	GPlace4.0 (MHz)	GPlace4.1 (MHz)	Improvement (%)
FPGA-1	263.2	**271.85**	3.29
FPGA-2	160.32	**162.90**	1.52
FPGA-3	115.28	**119.00**	3.10
FPGA-4	90.62	**92.60**	2.64
FPGA-5	47.81	**49.06**	2.65
FPGA-6	75.35	**77.83**	3.18
FPGA-7	61.51	**63.89**	3.17
FPGA-8	101.32	**103.65**	2.24
FPGA-9	57.59	**60.23**	5.24
FPGA-10	73.17	**74.84**	2.32
FPGA-11	68.43	**70.77**	2.85
FPGA-12	78.12	**81.16**	3.70

Table 5.30: GPlace4.0 and GPlace4.1 Maxmimum Frequency Comparison on 12 Benchmarks

Chapter 6

Conclusions and Future Work

This chapter concludes the thesis with a discussion of the achievements of the proposed framework in Section 6.1. Several directions for future work are proposed in Section 6.2.

6.1 Conclusion

In this thesis, several machine learning models were presented to predict the maximum frequency during placement. This commenced with the static and the dynamic features that were used in Models 1 and 2. It was observed that the models could predict F_{Max} with a high degree of accuracy in some groups while the accuracy was low in other groups. It was then shown that the model could be enhanced by combining the static and the dynamic features and adding timing features. A framework that had a high degree of accuracy for all of the groups was created. At each step, the effect of using simple and ensemble techniques was discussed, and it was observed that in most cases, ensembles were able to produce better results. A MAPE of 1.72% was obtained using the

best model. In the end, the ML framework was integrated into the placement engine of GPlace4.0. This combination could increase the maximum frequency that was achieved by an average of 3% for the 12 original benchmarks.

6.2 Future Work

Frequency prediction and its case studies are difficult and challenging tasks, and there are many other solutions that have not yet been considered. There are several aspects of this work that could be improved on in the future:

- This thesis used the predictions from ML model and a linear function to increase the criticality of paths and force the placer to increase the optimization effort. Still there are several approaches and functions that can be investigated to improve the results furthermore. Other probes like routing can be added to the case studies to make a balance between frequency and routability and decide and how far the timing optimization can be increased without reaching to an unroutable placement.

- Deep neural networks and CNNs could increase the accuracy of the model and the error generalization. These networks can engineer their own features based on the inputs, and find patterns and features that were not considered in this work.

- Predicting the maximum frequency of a circuit is a coarse grain prediction of the timing analysis of the design. Future works could include producing a complete analysis of all the timing paths and their post routing frequency.

Appendix A

Glossary

CAD	: Computer Aided Design
FPGA	: Field Programmable Gate Array
ASIC	: Application-Specific Integrated Circuits
FPGA	: Field Programmable Gate Array
FPGA	: Field Programmable Gate Array
HPWL	: Half Perimeter Wire Length
NP-hard	: Non Deterministic Polynomial Hard
STA	: Static Timing Analysis
ML	: Machine Learning
RF	: Random Forest
STD	: Standard Deviation
LUT	: Lookup Table
FF	: Flip Flop
DT	: Decision Tree

SVM	: Support Vector Machines
GPP	:General-Purpose Processors
NP-hard	: Non Deterministic Polynomial Hard
RTL	: Register Transfer Logic
SoC	: System on Chip
VHDL	: Very High Speed Integrated Circuit Hardware Description Language
VLSI	: Very Large Scale Integration
NCPR	: Net Cuts Per Region
WLPA	: Wire Length Per Area
MSE	: Mean Square Error
MAE	: Mean Absolute Error
RMSE	: Root Mean Square Error
MAPE	: Mean Absolute Percentage Error
BLE	: Basic Logic Element
CLB	: Configurable Logic Block

Bibliography

[1] J.-H. Yi, L.-N. Xing, G.-G. Wang, J. Dong, A. V. Vasilakos, A. H. Alavi, and L. Wang, "Behavior of crossover operators in nsga-iii for large-scale optimization problems," *Information Sciences*, vol. 509, pp. 470–487, 2020.

[2] G. Bonaccorso, *Machine learning algorithms*. Packt Publishing Ltd, 2017.

[3] F. Liu, A. Narayanan, and Q. Bai, "Real-time systems," 2000.

[4] K. Shahookar and P. Mazumder, "Vlsi cell placement techniques," *ACM Comput. Surv.*, vol. 23, no. 2, p. 143220, June 1991. [Online]. Available: https://doi.org/10.1145/103724.103725

[5] A. Al-hyari, H. Szentimrey, A. Shamli, T. Martin, G. Grewal, and S. Areibi, "A Deep Learning Framework to Predict Routability for FPGA Circuit Placement," *ACM Transactions on Reconfigurable Technology and Systems (TRETS)*, vol. -, no. -, pp. –, May 2021.

[6] D. Maarouf, A. Alhyari, Z. Abuowaimer, T. Martin, A. Gunter, G. Grewal, S. Areibi, and A. Vannelli, "A Machine-Learning Based Congestion Estimation for

Modern FPGAs," in *IEEE Int'l Conf. on Field Programmable Logic (FPL)*, Dublin, Ireland, August 2018, pp. 427–434.

[7] G. A. Seber and A. J. Lee, *Linear regression analysis*. John Wiley & Sons, 2012, vol. 329.

[8] C. Cortes and V. Vapnik, "Support-vector networks," *Machine learning*, vol. 20, no. 3, pp. 273–297, 1995.

[9] X. Wu, V. Kumar, J. R. Quinlan, J. Ghosh, Q. Yang, H. Motoda, G. J. McLachlan, A. Ng, B. Liu, S. Y. Philip, *et al.*, "Top 10 algorithms in data mining," *Knowledge and information systems*, vol. 14, no. 1, pp. 1–37, 2008.

[10] T. Chen and C. Guestrin, "XGBoost: A scalable tree boosting system," in *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ser. KDD '16. New York, NY, USA: ACM, 2016, pp. 785–794. [Online]. Available: http://doi.acm.org/10.1145/2939672.2939785

[11] T. K. Ho, "Random decision forests," in *Proceedings of 3rd international conference on document analysis and recognition*, vol. 1. IEEE, 1995, pp. 278–282.

[12] S. Džeroski and B. Ženko, "Is combining classifiers with stacking better than selecting the best one?" *Machine learning*, vol. 54, no. 3, pp. 255–273, 2004.

[13] M. Xu, G. Grewal, and S. Areibi, "StarPlace: A New Analytic Method for FPGA Placement," *Integration, The VLSI Journal*, vol. 44, no. 3, pp. 192–204, June 2011.

[14] T. Martin, D. Maarouf, Z. Abuowaimer, A. Alhyari, G. Grewal, and S. Areibi, "A

Flat Timing-Driven Placement Flow for Modern FPGAs," in *DAC Conference*, Las Vegas, USA, June 2019, pp. 1–6.

[15] G. Grewal and S. Areibi, "Guelph FPGA CAD Group: Xilinx Benchmarks," https://fpga.socs.uoguelph.ca/benchmarks/Xilinx.

[16] Xilinx, ""ISPD 2016 Routability-Driven FPGA Placement Contest"," http://www.ispd.cc/contests/16/ispd2016_contest.

[17] T. Martin, S. Areibi, and G. Grewal, "Effective Machine Learning Models for Predicting Routability During FPGA Placement," in *3rd ACM/IEEE Workshop on Machine Learning for CAD (MLCAD)*, Raleigh, NC, USA, September 2021, pp. –.

[18] L.-C. Chen, C.-C. Huang, Y.-L. Chang, and H.-M. Chen, "A learning-based methodology for routability prediction in placement," in *2018 International Symposium on VLSI Design, Automation and Test (VLSI-DAT)*, 2018, pp. 1–4.

[19] D. Marrouff, A. Shamli, T. Martin, G. Grewal, and S. Areibi, "A Deep-Learning Framework for Predicting Congestion during FPGA Placement," in *30th Int'l Conference on Field Programmable Logic and Applications*, Sweden, September 2020, pp. 138–144.

[20] E. C. Barboza, N. Shukla, Y. Chen, and J. Hu, "Machine learning-based pre-routing timing prediction with reduced pessimism," in *2019 56th ACM/IEEE Design Automation Conference (DAC)*, 2019, pp. 1–6.

[21] S.-Z. Zhang, Z.-Y. Zhao, C.-C. Feng, , and L. Wang, "A Machine Learning Frame-

work with Feature Selection for Floorplan Acceleration in IC Physical Design," in *JOURNAL OF COMPUTER SCIENCE AND TECHNOLOGY*, 2020, pp. 468–474.

[22] W.-T. J. Chan, K. Y. Chung, A. B. Kahng, N. D. MacDonald, and S. Nath, "Learning-based prediction of embedded memory timing failures during initial floorplan design," in *2016 21st Asia and South Pacific Design Automation Conference (ASP-DAC)*, 2016, pp. 178–185.

[23] S. Ganapathy, R. Canal, A. Gonzalez, and A. Rubio, "Circuit propagation delay estimation through multivariate regression-based modeling under spatio-temporal variability," in *2010 Design, Automation Test in Europe Conference Exhibition (DATE 2010)*, 2010, pp. 417–422.

[24] P. Cao, W. Bao, and J. Guo, "An accurate and efficient timing prediction framework for wide supply voltage design based on learning method," *Electronics*, vol. 9, no. 4, 2020. [Online]. Available: https://www.mdpi.com/2079-9292/9/4/580

[25] A. B. Kahng, U. Mallappa, and L. Saul, "Using machine learning to predict path-based slack from graph-based timing analysis," in *2018 IEEE 36th International Conference on Computer Design (ICCD)*, 2018, pp. 603–612.

[26] H. Makrani, F. Farahmand, H. Sayadi, S. Bondi, S. Dinakarrao, L. Zhao, A. Sasan, H. Homayoun, and S. Rafatirad, "Pyramid: Machine Learning Framework to Estimate the Optimal Timing and Resource Usage of a High-Level Synthesis Design," in *International Conference on Field Programmable Logic and Applications*, 2019, pp. 397–403.

[27] G. Wang, G. Stitt, H. Lam, and A. George, "Core-Level Modeling and Frequency Prediction for DSP Applications on FPGAs," *Int' Journal of Reconfigurable Computing*, vol. 2015, no. 2, pp. 1–20, April 2015.

[28] A. Nayak, M. Haldar, A. Choudhary, and P. Banerjee, "Accurate Area and Delay Estimators for FPGAs," in *Design Automation and Test in Europe Conference and Exhibition (DATE'02)*, 2002, pp. 862–869.

[29] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, *et al.*, "Scikit-learn: Machine learning in python," *Journal of machine learning research*, vol. 12, no. Oct, pp. 2825–2830, 2011.