ABSTRACT

QI, WEIYI. IC Design Analysis, Optimization and Reuse via Machine Learning. (Under the direction of Dr. Paul D. Franzon.)

Since the introduction of the Moore’s law in 1965, the integrated circuit industry has successfully managed over 50 years of exponential growth in design complexity and the transistor number has grown from thousands to billions on a single chip. Electronic design automation (EDA) tools are among the biggest factors that keep this growth trend and lead to the developments of cost- and energy-efficient robust electronic circuits and systems.

As technology node continues to scale down, the traditional EDA-based design methodology is challenged from many aspects. Firstly, the growing design complexity results in a significant increase in the computational cost and human labor for conducting thorough design analysis and optimization, both of which are keys to IC design successes. Secondly, the sophisticated underlying physics of advanced technology nodes make the modeling capability of the EDA tools questionable. In fact, most of the failures observed in qualification tests are direct results from such modeling issues, examples include mistuned analog circuits, signal timing errors, reliability problems, and crosstalk. The qualification failures in fabricated chips imply additional rounds of designs, known as design respins and it requires more efficient and reliable EDA tools to design high-yield circuits and systems aiming at the maximum utilization of the new technology and potentially eliminate the need for design respins.

In this work, we demonstrate how machine learning helps to alleviate the bottlenecks mentioned above. We particularly focus on the enhancement of simulation-based methodology for efficient design analysis, modeling, optimization, and yield estimation.

The fundamental idea for incorporating machine learning into the existing simulation-
based design methodology is to harness the statistical models’ capability of extracting information from the limited data set and make fast predictions about unobserved designs as well as accurately quantify the prediction uncertainty. The model can be used either as a direct surrogate of the expensive simulator or as a guide for the design decision-making process.

In this work, we demonstrate the efficacy of the proposed methodology through several circuit and system designs. Examples include the calibration of reliability-related degradations in mixed-signal circuits, fast configurations of the physical design flow, automatic analog circuit optimization and intellectual property (IP) reuse, and yield estimation of SRAM cells with low failure probability.
IC Design Analysis, Optimization and Reuse via Machine Learning

by
Weiyi Qi

A dissertation submitted to the Graduate Faculty of
North Carolina State University
in partial fulfillment of the
requirements for the Degree of
Doctor of Philosophy

Electrical Engineering

Raleigh, North Carolina

2017

APPROVED BY:

______________________________  ________________________________
Dr. Brian A. Floyd              Dr. William R. Davis

______________________________  ________________________________
Dr. Ranga R. Vatsavai           Dr. Paul D. Franzon
Chair of Advisory Committee
DEDICATION

To my parents, Baoxian Hao and Shuying Qi, who have taught me the virtues of commitment along the journey, with their love, wisdom, and encouragements.
BIOGRAPHY

Weiyi Qi was born in Xingtai, Hebei Province, China on December 27th, 1989. He received his Bachelor of Science (B.S.) degree in Electrical and Information Engineering from Dalian Jiaotong University in 2011, and the Master of Science (M.S.) degree in Electrical Engineering from North Carolina State University in 2013.

Weiyi joined Dr. Paul Franzon’s research group in February 2012, when he started his Ph.D. research on surrogate modeling for the acceleration of computationally expensive circuit simulation as a part of the self-HEALing mixed-signal Integrated Circuits (HEALICs) project funded by DARPA. In 2016, he became a student member of the Center for Advanced Electronics through Machine Learning (CAEML), with a research focus on machine learning aided design analysis, optimization, and reuse of analog intellectual properties.

Weiyi had two internships at Intel (formerly Altera) CAD group and Samsung Device Lab in 2015 spring and 2017 summer respectively.
ACKNOWLEDGEMENTS

No man is an island, entire of itself.

----------------------------------
John Donne

First of all, I would like to thank my advisor, Dr. Paul D. Franzon, for providing me with such incredible research and study opportunities. Dr. Franzon gives me the freedom to explore my ideas and the help when I am in the face of obstacles. His insights, enthusiasm, supports, and encouragements inspire me and guide me through this wonderful journey. To me, Dr. Franzon is more than an advisor in academics, but in life, too. Those wise words and deeds will become invaluable assets for me, forever.

I want to express my deepest appreciation to Dr. Floyd, who serves as a role model with his attitude and integrity to the academics; Dr. Davis, who gives me inspirations and suggestions through many of our discussions; and Dr. Vatsavai, from whom I acquired my rudimentary knowledge of machine learning and data mining. Also, my sincere gratitude goes to them for serving as my committee members.

I am also grateful to the following people and organizations, without whom I could not have made this accomplishment. These include Dr. Min Kang, who exposed me to the beauty of math and probability; Dr. Steve Lipa, whose help and advice are crucial at the beginning of my Ph.D. study; and my group fellows, Zhuo Yan, Jianchen Hu, Zhenqian Zhang, Wenxu Zhao, Weifu Li, Jong Beom Park, Josh Schabel, Zhao Wang, Nazia Zannat, Lee Baker, Kirti Bhanushali, Yi Wang, and Bowen Li, who have made special pieces of the beautiful memory of the past years. My courteous acknowledgment goes to DARPA, Raytheon, the CAEML center, and NSF for the supports of my research projects. I would also like to express my best feelings to the Samsung Device Lab and my colleagues for their inspirations during my internship.
Finally, I would like to give my most sincere gratefulness to my dear family. Oceans apart, but their love and support have always been the primary motivation.
CONTENTS

List of Tables .............................................................................................................. ix

List of Figures .............................................................................................................. x

Chapter 1 Introduction ................................................................................................. 1
  1.1 Motivation ........................................................................................................... 3
  1.2 Related Works I: Analog Optimization and IP Reuse ....................................... 6
    1.2.1 Knowledge-based approaches .................................................................. 7
    1.2.2 Optimization-based approaches ............................................................... 8
    1.2.3 Analog Optimization and IP Reuse Summary ......................................... 11
  1.3 Related Works II: Yield Estimation ................................................................. 13
    1.3.1 Crude Monte Carlo .................................................................................. 14
    1.3.2 Efficient Monte Carlo techniques .......................................................... 16
    1.3.3 Yield Analysis Summary ....................................................................... 22
  1.4 Dissertation Organization ............................................................................... 22
  1.5 Research Contributions .................................................................................... 24

Chapter 2 Surrogate Model Fundamentals ................................................................. 26
  2.1 Introduction to Surrogate Modeling .................................................................. 26
  2.2 Sampling Plan ................................................................................................... 28
    2.2.1 Latin Hypercube Sampling .................................................................... 31
  2.3 Model Construction ......................................................................................... 31
    2.3.1 Linear Regression ................................................................................... 33
    2.3.2 Probabilistic View of Linear Regression ................................................. 35
    2.3.3 Nonlinear Features and Basis Functions ................................................ 37
    2.3.4 Regularization ....................................................................................... 41
    2.3.5 Overfitting and Underfitting .................................................................. 44
  2.4 Model Selection and Cross Validation ............................................................ 46
  2.5 Adaptive Sampling ......................................................................................... 47
  2.6 Handling High Dimensionality with HDMR .................................................... 48

Chapter 3 Design Acceleration through Predictive Models ....................................... 52
  3.1 Case Study 1: VCDL Model ............................................................................. 53
    3.1.1 Aging Degradation and the NBTI effect ................................................. 53
  3.2 Case Study 2: Phase Rotator Model ............................................................... 58
  3.3 Case Study 3: Model-Based Physical Design ................................................ 62
  3.4 Summary .......................................................................................................... 70

Chapter 4 Machine Learning Based Analog IP Reuse .............................................. 73
4.1 Introduction ................................................................. 74
4.2 Overview of Proposed Approach .................................... 75
4.3 Design Analysis .......................................................... 76
  4.3.1 Design Parameter Screening ................................. 76
  4.3.2 Design Objective Analysis ................................. 78
4.4 Design Optimization ..................................................... 79
  4.4.1 Bayesian Optimization Algorithm .......................... 80
  4.4.2 Bayesian Linear Regression ......................... 81
  4.4.3 Inference with Bayesian Linear Regression .......... 84
  4.4.4 Comparison with Linear Regression .............. 85
  4.4.5 Gaussian Processes ............................................ 85
  4.4.6 Student-t Processes ......................................... 89
  4.4.7 Acquisition Functions ....................................... 91
4.5 Design Examples ........................................................... 94
  4.5.1 Test Case 1: 77GHz Balun ................................. 94
  4.5.2 Test Case 2: 77GHz PA Design .................... 99
4.6 Conclusions ................................................................. 103

Chapter 5  Predictive Model Enabled Efficient Yield Estimation .............. 104
  5.1 High Sigma Monte Carlo (HSMC) algorithm .................. 105
    5.1.1 Symbolic Regression and FFX ...................... 105
  5.2 Ensemble Models .................................................... 109
    5.2.1 Gradient Boosting ......................................... 110
    5.2.2 Random Forest ............................................. 112
  5.3 Implementation Details ............................................. 115
    5.3.1 Implementation Concerns .............................. 115
    5.3.2 Predictive Model Selection ....................... 116
    5.3.3 TAT Optimization .................................... 119
  5.4 Testing results ....................................................... 120
    5.4.1 Academic Test Case: Cantilever problem ........ 120
    5.4.2 6T SRAM Test Case .............................. 123
  5.5 Conclusion ............................................................... 126

Chapter 6  Conclusion and Future Work ..................................... 127

BIBLIOGRAPHY ................................................................. 133

APPENDIX ................................................................. 146
  Appendix A  Code and API Examples .............................. 147
    A.1 Surrogate Modeling Flow Code ............................. 147
    A.1.1 The Parallel Flowchart .............................. 147
LIST OF TABLES

Table 1.1 Comparative Study of Analog Optimization Techniques ................. 12
Table 1.2 Comparative analysis of Monte Carlo techniques ................. 23

Table 3.1 VCDL Inputs and Output Summary .................................................. 56
Table 3.2 Phase Rotator Inputs and Outputs Summary ................................ 59
Table 3.3 CORTEX Physical Design Parameter Summary ....................... 66
Table 3.4 Results of iteration 1 ................................................................. 69
Table 3.5 Physical Design Iterations ......................................................... 70

Table 5.1 Model parameter settings ............................................................. 116
Table 5.2 Prediction Speed Comparison (10^7 predictions) ....................... 117
Table 5.3 Cantilever Parameter Distribution ............................................. 117
Table 5.4 Prediction accuracy comparison .................................................. 118
Table 5.5 Cantilever performance comparison .......................................... 123
Table 5.6 6T SRAM Parameter Distribution ............................................. 124
# LIST OF FIGURES

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Figure 1.1</td>
<td>Levels of simulation tools</td>
<td>4</td>
</tr>
<tr>
<td>Figure 1.2</td>
<td>Illustration of Surrogate Modeling</td>
<td>6</td>
</tr>
<tr>
<td>Figure 1.3</td>
<td>Analog optimization tool categories</td>
<td>12</td>
</tr>
<tr>
<td>Figure 1.4</td>
<td>General reliability design procedure</td>
<td>14</td>
</tr>
<tr>
<td>Figure 1.5</td>
<td>General reliability design procedure</td>
<td>23</td>
</tr>
<tr>
<td>Figure 2.1</td>
<td>Key surrogate modeling stages</td>
<td>29</td>
</tr>
<tr>
<td>Figure 2.2</td>
<td>A 2-D example of latin hypercube sampling (10 points)</td>
<td>32</td>
</tr>
<tr>
<td>Figure 2.3</td>
<td>Handling nonlinear data with Linear Regression, created with the scikit-learn package [109]</td>
<td>39</td>
</tr>
<tr>
<td>Figure 2.4</td>
<td>Handling nonlinear data with Polynomial Regression, created with scikit-learn package</td>
<td>40</td>
</tr>
<tr>
<td>Figure 2.5</td>
<td>Regression using RBF model, created with SciPy package[63]</td>
<td>42</td>
</tr>
<tr>
<td>Figure 2.6</td>
<td>Under-fitting, desirable model, and over-fitting. Created with the scikit-learn package</td>
<td>44</td>
</tr>
<tr>
<td>Figure 2.7</td>
<td>Regularization for over-fitting. Created with the scikit-learn package</td>
<td>45</td>
</tr>
<tr>
<td>Figure 2.8</td>
<td>2-D example of Voronoi tessellation</td>
<td>49</td>
</tr>
<tr>
<td>Figure 3.1</td>
<td>Block diagram of the VCDL test infrastructure</td>
<td>55</td>
</tr>
<tr>
<td>Figure 3.2</td>
<td>The parallel simulation flow</td>
<td>57</td>
</tr>
<tr>
<td>Figure 3.3</td>
<td>Contour Plot of the VCDL Model</td>
<td>58</td>
</tr>
<tr>
<td>Figure 3.4</td>
<td>Block diagram the phase rotator design</td>
<td>60</td>
</tr>
<tr>
<td>Figure 3.5</td>
<td>The point-by-point simulation of the phase rotator phase response surface</td>
<td>61</td>
</tr>
<tr>
<td>Figure 3.6</td>
<td>Direct modeling lead to a smoothed response surface of the phase rotator phase</td>
<td>62</td>
</tr>
<tr>
<td>Figure 3.7</td>
<td>LOLA-Voronoi HDMR Modeling Flow</td>
<td>63</td>
</tr>
<tr>
<td>Figure 3.8</td>
<td>The Phase Rotator Phase Response Model</td>
<td>64</td>
</tr>
<tr>
<td>Figure 3.9</td>
<td>The Phase Rotator Magnitude Response Model</td>
<td>64</td>
</tr>
<tr>
<td>Figure 3.10</td>
<td>The Macro-Structure of the CORTEX SoC Platform</td>
<td>66</td>
</tr>
<tr>
<td>Figure 3.11</td>
<td>Trivial Effects of sinkMaxTran and maxSkew on Congestion</td>
<td>67</td>
</tr>
<tr>
<td>Figure 3.12</td>
<td>Trivial Effects of sinkMaxTran and maxSkew on Setup slack</td>
<td>68</td>
</tr>
<tr>
<td>Figure 3.13</td>
<td>Significant Effects of sinkMaxTran and maxSkew on Hold slack</td>
<td>68</td>
</tr>
<tr>
<td>Figure 3.14</td>
<td>Congestion vs. Layers (CLKper = 10, density = 0.6)</td>
<td>69</td>
</tr>
<tr>
<td>Figure 3.15</td>
<td>Final design details</td>
<td>71</td>
</tr>
<tr>
<td>Figure 3.16</td>
<td>Snapshot of the final routed design</td>
<td>72</td>
</tr>
<tr>
<td>Figure 4.1</td>
<td>Proposed work flow</td>
<td>76</td>
</tr>
<tr>
<td>Figure 4.2</td>
<td>(a) Balun schematics and (b) parameter screening</td>
<td>95</td>
</tr>
</tbody>
</table>
Figure 4.3  Design goal distribution and correlation ................................. 96
Figure 4.4  Balun optimization comparison .............................................. 97
Figure 4.5  (a) Manual balun design in 8XP; (b) Balun redesign in 8XP; (c) Balun reuse in 9HP ................................................................. 98
Figure 4.6  Power amplifier schematics ...................................................... 99
Figure 4.7  PA Design parameter screening .............................................. 100
Figure 4.8  PA optimization comparison .................................................. 101
Figure 4.9  (a) Manual PA Design in 8XP; (b) PA redesign in 8XP; (c) PA reuse in 9HP; (d) Result comparisons (*Simulated at 80GHz*) ...................... 102

Figure 5.1  Illustration of the symbolic regression tree ............................... 107
Figure 5.2  Sample distribution of iteration 1, 2, 4, and 8 .......................... 121
Figure 5.3  Convergence plot of the Cantilever example ........................... 122
Figure 5.4  SRAM schematics ................................................................. 124
Figure 5.5  Convergence plot of the 6T-SRAM example ............................ 125
Integrated circuit (IC) designs depend on sophisticated parameter tuning and decision making with the help of software tools, commonly known as the computer aided design (CAD) or electronic design automation (EDA) tools. Over the past few decades, the advancements in computing power has given rise to an explosion in the capability of the EDA software that guarantees the IC industry to keep up with the Moore’s law [105]. Among all the EDA tools, simulation is the key to drive the development of low-cost, safe and robust circuits and systems. Circuit simulations happen in different levels, where higher level ones
obscure the lower level details in order to trade off the overall performances of simulation speed and accuracy[19]. From bottom to top, these abstraction levels include:

1. **Physics level:**
   At device physics level, the goal is to build technology CAD (TCAD) models of the semiconductor fabrication and device operations. The former one is called *process TCAD*, while the latter *device TCAD* [78]. Both process and device TCAD models have high fidelity at the physics level, but it is too expensive to be used directly in circuit simulations, therefore, TCAD also produces compact models that capture the electrical behavior as mathematical equations without resorting to the underlying physics. A well known example of compact models is the SPICE transistor model.

2. **Circuit level:**
   Circuit level simulators analyze transistors, wires, capacitors, resistors, and their interconnectivity[19], then model the schematics or layout behaviors by solving differential linear equations formed from operation principles such as the Kirchhoff’s laws (KVL and KCL). Circuit level simulators produces accurate waveforms that resembles the real world operation. Accurate as it is, this technique can be computationally intensive, especially for larger designs, so it is usually used in the most critical blocks of a system, often the analog and mixed-signal subsections.

3. **Logic level:**
   Logic level simulations steer away from the continuous data and attempt to abstract the behavior at the boolean level (1/0/X). Further assumptions are made to facilitate the simulation speed, such as ignoring the wire resistance and the coupling effects.
At a lower logic level, each transistor is treated as a switch that has either 1 (on) or 0 (off) states, this is called the switch-level simulation; at a higher logic level, groups of transistors, resistors, and capacitors are further abstracted to logic gates, flip-flops, XOR, for instance, this is known as the gate-level simulations. Logic level simulators makes it possible to efficiently study the behaviors of more complicated systems.

4. Behavioral level:

Behavioral level simulations are also known as the functional level simulations. These simulations model the architectural operations and are accurate at cycle- or interface-level[78]. Behavioral level simulations are useful as they provide an overall picture of how design and control flows work and interact, thus making rapid prototyping and path-finding possible. But as most details are hidden, behavioral simulations could easily lead to infeasible designs or tedious lower-level implementations.

By representing the simulation levels as a stack structure, an illustration is shown in Figure 1.1 below.

1.1 Motivation

Hierarchical simulation abstraction levels trade off simulation speed and accuracy by gradually hiding away the details from lower levels. Still, simulation-based designs are reaching their bottlenecks as the IC design complexity grows. On one hand, the overhead in computational cost and the simulation speed has increased dramatically; on the other hand, the simulation accuracy and model capability are severely challenged by the complicated

---

13D process model plot from [65]
Figure 1.1 Levels of simulation tools
circuit behaviors. As a consequence, the design process based on simulation tools become extremely inefficient, and there is an urgent need to bridge the different levels and create models that are fast enough but without severely impair the accuracy.

We can achieve this goal using surrogate models, also known as meta-models or response surface models (RSM)\(^2\). Such models seek to approximate the real and expensive simulation responses and make predictions of the unseen by augmenting the information given by a limited number of observed data. A fundamental assumption of this approach is that the surrogate models are orders of magnitudes faster while still being accurate enough to use\([33]\).

We show the general idea of surrogate models in Figure 1.2. Essentially, surrogate models stems from the family of predictive models in supervised machine learning. The main distinction of surrogate models from other supervised machine learning models is the ability to cope with the scarcity of training data. Generally speaking, surrogate models obscure the underlying physics and seek to represent the input-output relationships using a "learned" mapping function. The input space is strategically sampled so as to minimize the data consumption and improve the modeling efficiency while still ensuring enough information being retrieved regarding the underlying "true" relation.

Surrogate models proves to be useful in various fields, such as aerodynamics\([71]\), structural reliability\([38]\), mechanical designs\([120]\), I/O circuits\([142]\), semiconductor device modeling\([137]\), microwave circuit optimiation\([6]\), etc.

In this work, we try to harness the advantages of speed-accuracy trade-off associated with surrogate models to tackle design exploration, optimization, and yield analysis of

\(^2\)In the rest of this paper, we will use the terms "surrogate models", "predictive models", "metamodels" and "response surface models" interchangeably.
IC and system designs. In particular, this dissertation emphasizes on automatic analog intellectual property (IP) reuse and reliability analysis of rare-event circuit failures. In the rest of this chapter, we will conduct a literature review of the existing works on these two topics. For the sake of brevity, interested readers are referred to [102] for a discussion on how predictive models have been used in other EDA fields, such as circuit level and architecture level path-findings.

1.2 Related Works I: Analog Optimization and IP Reuse

In recent years, considerable research efforts in both academic and industrial communities are devoted to the computer-aided solutions of designing analog circuits. Aiming at full functionality, these tools often cover the entire workflow of analog synthesis, including topology selection, performance optimization, and layout generation. In this work, however, we choose to focus on the core problem of performance optimization for IP reuse, which
consists re-optimization in the same process nodes (called re-design) and migration to new nodes (or reuse). Note that, by "reuse", we assume the identical topology is maintained in both cases. We will discuss the most significant efforts in related fields in the next sections. Briefly speaking, these techniques can be broadly categorized into knowledge-based approaches and optimization-based approaches[86].

1.2.1 Knowledge-based approaches

The main purpose of knowledge-based approaches is to encapsulate the designers’ knowledge, whether it be building a pre-design plan with design equations and a design strategy that produces the component sizing[7] or representing the design procedure as a set of flowcharts and converting the intrinsic dependencies of device parameters into graphs[46] to guide the design and optimization, hence two main groups of knowledge-based approaches can be found in literature: design plan driven methods and operating point driven methods.

A) Design Plan Driven

This method characterizes a complete design plan that describes how the circuit components need be sized to reach the design specifications. Analytical or empirical equations are often used to explore the design space and trade off performances.

Representatives of design plan driven knowledge-based approaches includes [24, 30, 48, 52, 79]. The main advantage of this method is that once the execution plan is defined, the speed for sizing is very efficient and the solution quality only depends on the evaluator’s accuracy. However, there are two significant drawbacks: (1) the overhead of defining and reforming the design plans[51], and (2) the limitation to only a few sets of circuit topolo-
gies[48]. Moreover, though satisfying the design specifications, there is no guarantee in finding the optimum solution using the design plan drive approaches[89].

B) Operating Point Driven

More recently, works on the knowledge-aware analog synthesis and IP reuse are proposed in [57, 59, 60]. The essential foundation of this approach is on the operating-point driven formulation of analog CAD[46] and derived from traditional handcrafted analog circuit design method, where the operating points are calculated first, and then the attributes (such as aspect ratios) can be determined from the operating points by inverting the analytic equations. The CAIRO+ platform[56] uses bipartite graphs to express the operating point dependency. Together with a group of operators for inverting the BSIM compact models, this framework is able to determine the transistor aspect ratios. The advantage of such knowledge-aware operating point drive approach is that designer can maintain insights of the impacts of design variables (voltages, currents) on performances thus enhance the efficiency of circuit reuse in the same technology. However, the inverse operators need to be updated or re-developed to support different compact models[55]. Therefore, it is less generally applicable and also results in a long preparation time, similar to other knowledge-based approaches.

1.2.2 Optimization-based approaches

Optimization-based approaches translate the analog optimization and IP reuse problem to the minimization (or maximization) of objective functions, which is then solved by numerical methods or optimization algorithms. Instead of using design plans, optimization algorithms interpret the design process as an iteration in which design variables are tuned
till the design meets particular specifications. Nowadays, optimization-based approaches are widely accepted and used[117], mainly due to the automaticity characteristic that keeps a human out of the loop. To form this automatic loop, optimization algorithms are paired with performance evaluators that evaluate the quality of the design. Three main types of performance evaluator are used, i.e. analytical equation, circuit simulator and behavioral models[7].

A) Analytical equations

The analytical equations are derived either manually or with automated tools. Relevant works include [11, 40, 42, 49, 69, 88, 91, 95, 107] . Analytical equations are conceptually mathematical representations of the circuit blocks and are known for their short evaluation time which is a significant advantage in design space exploration and performance optimization. However, despite recent advances in symbolic circuit analysis[87], not all design characteristics are captured with equations. One reason is that there is a dilemma that closed-form analytical equations call for simplifications, which in turn results in inaccuracy and incompleteness in analytic equations. Using inaccurate or incomplete evaluators will impose severe restrictions to the final optimized circuit performance.

Asides from the methods above, [1, 22] proposed to express analog designs and optimization tasks in posynomial forms. As the posynomial expressions are intrinsically convex, the global solution can always be found regardless of where the starting point is[11], thus can be solved efficiently with geometric programming techniques. Nevertheless, reformulating accurate device models in posynomial forms is challenging, as the author indicated: “performance specifications and objectives that can be handled are far more restricted than any of the methods described above.”[11]
B) Circuit simulator

This method incorporates circuit simulators in the optimization loop to evaluate the performances. [72, 73, 84, 86, 95, 106, 110, 127] cover various uses of circuit simulators as performance evaluators. Clear advantages over the equation-based approach are (1) the evaluation accuracy and flexibility for different kinds of analog design blocks, and (2) it also allows reliability analysis such as the Monte Carlo and aging degradation analysis [112], both with minimal setup time.

However, in spite of the clear advantages of simulator-in-the-loop methods, the computational cost of circuit simulators prohibits its further adaptability, as common optimization algorithms (such as evolutionary algorithms) on analog circuit requires hundreds or even thousands of function evaluations to achieve optimal designs [84].

C) Behavioral models

To address the accuracy issue in equation models and computational drawback in the simulator-in-loop method, learning-based methods provide a promising solution to enhance the efficiency of optimization-based methodology. The idea is to build behavioral models from a set of training data which are sampled by executing circuit or system simulations and then use machine learning techniques to model the input-output relationship mapping. In [85], Liu et al. proposed data-mining based macro-modeling techniques for large analog design space exploration, and demonstrated the clear optimization efficiency. Recent works regarding behavioral model-based techniques on analog optimization are presented in [3, 23, 85, 133]. For example, [3] used neural-fuzzy models with evolutionary optimization strategies and [23] developed support vector machine (SVM) models to represent the analog circuit behavior in the design space. In [133] a neural network macro-
model was trained to be used for analog circuit synthesis, the model can substitute for full
SPICE simulation and the inclusion of design rules for sizing and operating points also
substantially shrank the design space and ensured correct functionality.

Behavioral models are comparable to full circuit simulations in accuracy and have the
speed on a par with analytic equations. This characteristic makes it particularly favorable
for optimization-based approaches[133]. Successful applications of model-based analysis
have been demonstrated in [135, 137, 138, 142], which shows that behavioral models can
also characterize circuit variability and reliability to improve design robustness. Moreover,
in the case of analog IP reuse, behavioral models can additionally help alleviate the IP
bottleneck, in which surrogate models act as substitutes of the detailed schematics or
layouts.

To train such accurate predictive models, a set of training samples from the high fidelity
evaluators, such as SPICE simulators, are required. The number of training samples reflects
the trade-off between computational cost and model performance: for more complicated
circuits with higher dimensionality, the computational cost of maintaining a desirable
accuracy increases exponentially with the number of free parameters. This phenomenon is
known as “the curse of high dimensionality” [33].

1.2.3 Analog Optimization and IP Reuse Summary

The summarization of different analog optimization and IP reuse techniques in presented
Figure 1.3 with a comparative analysis in Table 1.1. In the table, "X", "-", and "✓" stand for
"good", "moderate", and "bad", respectively. From this table, we notice that model-based
approach shows no significant drawbacks.
Figure 1.3 Analog optimization tool categories

Table 1.1 Comparative Study of Analog Optimization Techniques

<table>
<thead>
<tr>
<th></th>
<th>Knowledge-based</th>
<th>Optimization-based</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Design Plan</td>
<td>OP Driven</td>
</tr>
<tr>
<td>Setup Time</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Design Time</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Accuracy</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Robustness</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>
1.3 Related Works II: Yield Estimation

Yield estimation is also known as reliability analysis. It is an essential part of almost any aspect of engineering design. Specifically, for IC designs, the designers need to take into consideration of two kinds of variations: the variations during manufacturing and the uncertainty during operations. The former one is known as the process variations (P) while the latter includes voltage (V) and temperature (T). Reliability analysis for IC design thus requires the identification of chances that the system enters a suboptimal situation under PVT variations, or equivalently estimating the probability of failure ($\alpha_f$).

Reliability aware design is critical in all IC design tasks, including radio-frequency (RF), analog, mixed-signal circuits, I/O designs, and memory, etc. The reliability aware design flow usually consists of corner-based design followed by a Monte Carlo verification step[93], with the flow diagram shown in Figure 1.4:

However, there are two main drawbacks associated with this procedure: corner models, despite their appealing speed, usually lead to pessimistic designs that cause the over-design problem; while the Monte Carlo step gives a better estimation of the final yield, but at the cost of lengthy simulation and massive computational power.

Many alternatives are proposed in the literature, including sophisticated designs of experiments (DoE), improved statistical estimation, and surrogate model enabled Monte Carlo techniques. In the next sections, we will first discuss how crude Monte Carlo method works and its bottleneck in estimating rare events, and then conduct a brief review of the existing techniques for comparative analysis.
1.3.1 Crude Monte Carlo

Due to the black-box nature of the distribution of performance metrics, designers need to rely on simulations to assess the design failure probability, which is typically done using the Monte Carlo approach. Essentially, Monte Carlo method involves repeating the simulation many times with different design or environment parameter settings and approximating the failure rate by countering the observed failures against the total number of simulations conducted.

Consider a $d$-dimensional random vector $X$ distributed according to $P_X$ with a probability density function (PDF) $p_X$, these are the inputs (design parameters and / or PVT) to the circuit / system under simulation, which we denote as $f : \mathbb{R}^d \rightarrow \mathbb{R}$. Note, however,
for simplicity, we only consider $f$ as a scalar function, but it can be readily generalized to multiple-output cases. Suppose the system fails when $f(X) > u$, where $u \in \mathbb{R}$ is called the \textit{threshold} determined from the design specifications. The failure region is therefore:

$$
\Gamma = \{ x \in X | f(x) > u \}. \quad (1.1)
$$

And the probability of failure:

$$
\alpha_f = P_X(\Gamma) = \int_{\Gamma} p_X(x) dx = \int_X 1_{\Gamma}(X)p_X(x) dx. \quad (1.2)
$$

where $1_{\Gamma}(X)$ is an indicator function:

$$
1_{\Gamma}(X) = \begin{cases} 
1 : x \in \Gamma \\
0 : x \notin \Gamma 
\end{cases} \quad (1.3)
$$

With crude Monte Carlo, $\alpha_f$ is approximated as:

$$
\hat{\alpha}_f = \frac{1}{N} \sum_{i=1}^{N} 1_{\Gamma}(X). \quad (1.4)
$$

According to the law of large numbers\cite{58}, this estimation is an unbiased estimate that converges to the real probability as $N$ increases. One usually use the relative error to quantify the estimation efficiency\cite{96}. The relative error (also known as the relative deviation) is:

$$
RE(\hat{\alpha}_f) = \frac{\sigma_{\hat{\alpha}_f}}{E(\hat{\alpha}_f)}. \quad (1.5)
$$

As $\hat{\alpha}_f$ is unbiased, we have $E(\hat{\alpha}_f) = \alpha_f$, and using the properties of sampling distribution
of proportion, we have $\sigma_{\hat{\alpha}} = \sqrt{\frac{\hat{\alpha}(1-\hat{\alpha})}{N}}$, therefore,

$$RE(\hat{\alpha}) = \frac{\sqrt{\frac{\hat{\alpha}(1-\hat{\alpha})}{\sigma_{\hat{\alpha}}}}}{N} = \sqrt{\frac{1 - \hat{\alpha}}{\hat{\alpha}N}}.$$  

(1.6)

To illustrate the inefficiency of crude Monte Carlo method under computational budget when estimating a small probability of failure, suppose the estimation calls for 10% relative error, for estimating $\alpha = 10^{-3}$, the total sample number needed from a crude Monte Carlo simulation is:

$\frac{100}{\hat{\alpha}} = 10^5$.

Another intuitive view of the inefficiency is that most simulations are wasted because they don’t catch the failure cases. Again, suppose we want to verify $\alpha = 10^{-3}$, on average, 1000 simulations are needed in order to catch a single failure simulation.

### 1.3.2 Efficient Monte Carlo techniques

Efficient reliability analysis for small probabilities of failure is indispensable when designing robust circuits and systems and has drawn significant attentions from people with different backgrounds. The approaches are usually sorted into three main categories[8]: geometric approximation in the distribution space, variations of the Monte Carlo approach, and surrogate model assisted methods. In this section, we will review the representative works in each subcategory.
1.3.2.1 Geometric Approximation

Geometric approximation methods aim at approximating the contour of the failure region \((\Gamma)\) geometrically with simpler shapes \(\hat{\Gamma}\). An approximation of the integration in Equation (1.4) is:

\[
\hat{a}_{f(Geo)} = P_X(\hat{\Gamma}) = \int_X 1_f(X)p_X(x)dx.
\]  

An apparent advantage of this approach is that once we form \(\hat{\Gamma}\), no further evaluation is needed to estimate \(a_f\) as it is calculated using probability calculus. In other words, all function evaluations or simulations are spent in finding the contour approximation.

An example of the geometric approximation is the first/second order reliability methods (FORM/SORM) which are prevalent in structural safety literature [83]. The typical steps for conducting FORM/SORM involves four steps:

1. **Input distribution transformation:** The input distribution is mapped from \(P_X\) into a standard multivariate normal distribution \(V\).

2. **Search for the Most Probable Point (MPP) in \(V\):** The MPP is the point on the failure boundary that is closest to the distribution center (So it has the highest probability of failure among all the points residing on the failure boundary).

3. **Geometrically approximating the failure boundary:** This step can be regarded as a first-/second-order expansion of the failure boundary with respect to the MPP, in which one fits a tangent hyperplane (for FORM) or a paraboloid (for SORM) and use it to estimate
the true failure region.

(4) **Calculating the probability of failure:** With the approximated failure boundary, the calculation of failure probability in $V$ is simply done using the standard normal pdf.

It is apparent that the success of this category of methods depends on the contour approximation quality. Several factors limit this quality, however, such as the nonlinear transformation in Step (1) or multimodality issues, where more than one MPP exists.

### 1.3.2.2 Variants of Monte Carlo Method

Many variants are proposed to overcome the inefficiency of crude Monte Carlo method. Among all the techniques, we will discuss three most representative ones, including *quasi Monte Carlo methods*, *importance sampling*, and *subset simulation*.

(1) **Quasi Monte Carlo methods**

These methods are also known as low-discrepancy sampling and base themselves on variance reduction techniques[116]. The main idea is to generate samples in the distribution space with a better spread, so as to achieve the same accuracy of crude Monte Carlo simulation with fewer samples. For example, [50] and [27] demonstrated the use of Latin Hypercube sampling and stratified sampling respectively. However, it doesn't solve the root problem regarding Monte Carlo simulation: for a one-in-a-billion failure case, one still needs, on average, 1 billion simulations to capture a single failure point.

(2) **Importance sampling**

The core of importance sampling is to strategically modify the sampling distribution so that the probability density function (PDF) can be shifted towards the failure region. As a consequence, more points are drawn in $\Gamma$. 
Let $h$ be a proposal PDF, the failure probability $\alpha$ is written as:

$$
\alpha_f = \int_X 1_\Gamma(X)p_X(x) \, dx
= \int_X 1_\Gamma(X)\frac{p_X(x)}{h(x)} \, h(x) \, dx
= E_h\left(1_\Gamma(X)\frac{p_X(x)}{h(x)}\right).$$

(1.8)

where $E_h$ denotes the expectation with respect to the proposal PDF $h$. Now, with importance sampling:

$$
\hat{\alpha}_{f(IS)} = \frac{1}{N} \sum_{i=1}^{N} 1_\Gamma(X)\frac{p_X(x)}{h(x)}.
$$

(1.9)

The primary challenge for importance sampling is to effectively determine the optimal proposal density function $h^*$. According to [96], this optimal is reached by minimizing the variance of $\hat{\alpha}_{f(IS)}$:

$$
h^* = \arg\min Var(\hat{\alpha}_{f(IS)}) = \frac{1_\Gamma(X)p_X}{\int_X 1_\Gamma(X)p_X \, dx} = \frac{1_\Gamma(X)p_X}{\alpha}.
$$

(1.10)

Notice that $h^*$ in turn depends on the unknown failure region $\Gamma$, so the minimization problem is ill-posed. To overcome this dilemma, [116] introduced the adaptive importance sample technique that involves generating intermediate Monte Carlo simulations with proposal densities $h_i$, $i = 1, ..., T$ to sequentially approach $h^*$, either parametrically or non-parametrically. In parametric adaptive importance sampling, one uses a vector of parameters to characterize the intermediate proposal functions, while for non-parametric ap-
proach, instead of estimating the density parameters, the density function itself is estimated, such as with the Gaussian kernels[43]. In both methods, the quality of the approximation is measured with the Kullback–Leibler divergence[96].

Importance sampling techniques is the state-of-the-art rare-event simulation algorithm in lower dimensional cases. However, as dimensionality increases, finding good proposal distribution functions becomes nontrivial, and therefore a screening step is necessary to identify the most important features and reduce the dimensionality.

(3) **Subset simulation**

Instead of approximating the underlying distribution function, as with importance sampling, subset simulation instead partitions the performance domain by dividing the rare event into a series of nested less rare events, the failure probability is then a product of the nested conditional probabilities.

Precisely, one builds a sequence of increasing thresholds \( \{u_i\}_{i=1,...,T} \), such that \(-\infty < u_1 < u_2 < ... < u_T = u\). This correspond to a decreasing sequence of failure regions \( \{\Gamma_i\}_{i=1,...,T} \), where \( X \supseteq \Gamma_1 \supseteq \Gamma_2 \supseteq ... \supseteq \Gamma_T = \Gamma \).

The probability of failure is therefore:

\[
\alpha_{f(SS)} = P_X(\Gamma) = P_X(\Gamma_1) \prod_{t=1}^{T-1} P_X(\Gamma_{t+1}|\Gamma_t) = \prod_{t=1}^{T} p_t
\]

(1.11)

For each \( p_t = P_X(\Gamma_{t+1}|\Gamma_t) \), suppose we generate samples according to \( P_X(.|\Gamma_t) \), then with probability \( p_t \) the samples will fall into \( \Gamma_{t+1} \). These samples are called the **partial samples**

20
in subset simulation literature. In each iteration $t$, we keep the samples located in $\Gamma_{t+1}$ and generate an i.i.d sample using the sequential Monte Carlo method\[9\].

Subset simulation uses the divide-and-conquer technique and efficiently decomposes the complicated situation into a sequence of easier-to-solve problems. However, in each decomposed step, even though the overall required number of samples is significantly smaller than that of the crude Monte Carlo methods, the conditional probability is usually determined by Markov Chain Monte Carlo (MCMC) simulations that still requires a relatively high simulation cost. This makes subset simulation unappealing characteristic when handling expensive evaluations, such as radio-frequency circuits and electromagnetic designs.

### 1.3.2.3 Surrogate model assisted methods

In the first part of related works, we have discussed the advantages of surrogate/behavioral models for optimizing expensive simulation-based analog designs. Indeed, surrogate models also help Monte Carlo simulations: by modeling the expensive system with a cheaper-to-evaluate surrogate, the reliability estimation is done either with a direct substitution or guided by the surrogate.

When surrogate model is used as a direct substitution, it is called the response surface Monte Carlo method. Earlier examples include\[13, 29, 31\], and some recent developments are demonstrated in \[39, 66, 139\]. In \[108\] artificial neural network model is adopted, and \[54\] utilizes the SVM model. The direct substitution method is a simple and straightforward approach for incorporating surrogate models in the reliability estimation loop, but as with the analog optimization problem, it calls for high-quality models to reach satisfactory
results. Therefore, when handling higher dimensional or nonlinear response surfaces, the performances of direct substitution is in doubt.

Instead of using surrogate models directly, one can also use the model to guide the sampling steps. For example, [122] used a decision tree to block unnecessary simulations of an SRAM cell, and [8] uses Gaussian process model for sequential Monte Carlo simulation. In [9], Gaussian processes are incorporated in subset simulation, resulting in a technique called the Bayesian subset simulation that avoids the cost of MCMC for conditional probability estimation at each step and instead use the information given by the Gaussian process predictions. In [93], a symbolic regression model is trained to sort the simulation sample sequence and perform data mining in the massive Monte Carlo samples.

### 1.3.3 Yield Analysis Summary

We summarize the categories of yield analysis techniques in Figure 1.5 and also present a comparative study in Table 1.2. In the table, "XX" means extremely prohibitive, "X" and "✓" indicate bad and good, respectively, while blanks denotes "moderate" or hard to evaluate. Again, we notice that model assisted Monte Carlo simulation is the most promising solution, especially when computational resource is under tight budgets.

### 1.4 Dissertation Organization

In the following chapters, we begin our discussion with an overview of the machine learning and surrogate model fundamentals, including sampling plan, model constructions, and model selections in Chapter 2. Then, in Chapter 3, we use predictive models in three
Figure 1.5 General reliability design procedure

Table 1.2 Comparative analysis of Monte Carlo techniques

<table>
<thead>
<tr>
<th></th>
<th>Crude MC</th>
<th>Quasi MC</th>
<th>FORM/SORM</th>
<th>Importance Sampling</th>
<th>Subset Simulation</th>
<th>Direct Model</th>
<th>Model Assisted</th>
</tr>
</thead>
<tbody>
<tr>
<td>Computational Cost</td>
<td>XX</td>
<td>X</td>
<td>✓</td>
<td></td>
<td></td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Estimation Accuracy</td>
<td></td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td>X</td>
<td>✓</td>
</tr>
<tr>
<td>High Dimensionality</td>
<td>✓</td>
<td>✓</td>
<td>X</td>
<td>X</td>
<td>✓</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>Disjoint Failure Region</td>
<td>✓</td>
<td>✓</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>Nonlinearity</td>
<td>✓</td>
<td>✓</td>
<td>X</td>
<td></td>
<td>✓</td>
<td></td>
<td>✓</td>
</tr>
</tbody>
</table>
design examples, including a voltage-controlled delay line (VCDL), a phase rotator, and a physical design of the CORTEX processor. In Chapter 4, we present the automatic analog design analysis, optimization, and reused flow with the core algorithm called Bayesian optimization. We continue our discussion on how ensemble models enhance the efficient yield estimation of SRAM circuits in Chapter 5. In the final chapter, a summary and an outlook of future work are presented.

1.5 Research Contributions

The main contributions of this dissertation include:

1. **Implemented a parallel model construction flow to facilitate the development of aging-induced degradation calibration algorithm.** *(Chapter 3)*

   Sequential surrogate modeling flow typically adds one new sample at each iteration, in this parallel flow, we modified the sampling criteria so that more than one sample is requested each round and implemented the multi-server parallel infrastructure to greatly expedite the model construction process. The flow is demonstrated with a voltage-controlled delay line and a phase rotator design and is > 10X speedup, compared with the original flow in [141].

2. **Proposed and implemented the High Dimensional Model Representation (HDMR) modeling flow for handling high-dimensionality in behavioral models.** *(Chapter 3)*

   When modeling the phase rotator problem, traditional predictive models tend to destroy the desired response surface by smoothing out the nonlinear regions. We introduced the used of HDMR technique and decomposed the complicated problem into an additive set
of lower dimensional ones and was able to preserve the response surface trait to test the robustness of the calibration algorithm.

3. Proposed and implemented an automatic flow for analog circuit design analysis, optimization, and intellectual property reuse. *(Chapter 4)*

Analog redesign and reuse between technology nodes are design-expertise intensive. We advocate the use of Bayesian optimization along with Student-\(t\) processes to efficiently produce high-quality designs. Moreover, this is the first application of Student-\(t\) model based Bayesian optimization in engineering designs.

4. Improved the HSMC algorithm with ensemble models for the estimation of rare-event failures in SRAM. *(Chapter 5)*

We enhanced the high-sigma Monte Carlo (HSMC)[93] algorithm using ensemble learning models. The resulted algorithm proves to converge faster than the original one, and we demonstrate its efficacy with an industrial SRAM test case.
In this chapter, we will introduce the fundamental concepts in surrogate models, and lay the foundation for in-depth discussions of different predictive modeling techniques along with their applications in the follow-on chapters.

2.1 Introduction to Surrogate Modeling

Recall that the goal of surrogate modeling is to train a mapping function using limited observed data points from the costly simulator or experiments, so this mapping function
can be used as a surrogate to represent the original process. Precisely, suppose the original expensive model is denoted as \( y = f(x) \), where \( x \) is the input vector, \( f \) is the true function, and \( y \) is the output. We wish to "learn" an estimated function \( \hat{f}(x) + \epsilon(x) \) of the original mapping from a small observed dataset \( D \) generated from \( f \). This estimation contains two parts: \( \hat{f}(x) \) is the expected value from the surrogate model and \( \epsilon(x) \) quantifies the uncertainty of the estimation.

This is indeed a regression problem in the statistical learning realm, in which a good paradigm for selecting the estimator \( \hat{f} \) is by solving the Tikhonov regularization problem[113]:

\[
\min_{f \in \mathcal{H}} Z(\hat{f}) = \frac{1}{N_s} \sum_{i=1}^{N_s} L(y^{(i)} - \hat{f}(x^{(i)})) + \lambda \int_{x} \|f^{(k)}(x)\|_{\mathcal{H}} \, dx. \tag{2.1}
\]

where \( N_s \) denotes the number observed data points; \( \mathcal{H} \) is the set of all model candidates; \( L(x) \) is the loss function that quantifies the differences between the true model and \( \hat{f} \) estimation; \( \lambda \) is the regularization parameter and \( \hat{f}^{(i)}(x) \) is the value for the \( m \)-th derivative of \( \hat{f} \) at \( x \).

Of the two additive components in Equation 2.1, the first term represents the "closeness" between \( \hat{f} \) and the true \( f \), while the latter denotes the "smoothness" of \( \hat{f} \), and these two terms are balanced with \( \lambda \). Intuitively, this minimization problem favors smooth functions that are close to the real function in the hypothesis family \( \mathcal{H} \). Also note that increasing \( \lambda \) leads to preferring "smoothness" over "closeness" and vice versa. With this guideline, we can conduct more detailed analysis in the later chapters.

The key stages for constructing a surrogate model includes:

1. Design of Experiments (DoE):
In this step, we create the initial sampling plan that ensures design space coverage with limited number of samples.

2. **Model Construction:**
Using the available samples, we train a mapping function (the surrogate model) that fits the observed data points well without losing its capability of predicting unobserved ones.

3. **Adaptive Sampling:**
This stage identifies the design space regions that has suboptimal predictive performance (due to nonlinearity, for instance), and sequentially add additional samples in those regions.

4. **Model validation:**
In this steps, we evaluate the final model and validate its quality.

The flowchart is shown in Figure 2.1. We will discuss each of these procedures in detail in the upcoming sections.

### 2.2 Sampling Plan

Compared to other supervised learning regression models, surrogate models emphasize on data efficiency, as the amount of samples is severely limited by the computational expense. Therefore, it is crucial to have a high-quality initial sampling plan or design of experiment (DoE).
As discussed in the introduction, we would like to define a sampling plan \( X = \{ x^{(1)}, x^{(2)}, \ldots, x^{(m)} \} \subset D \), and execute the expensive simulations/experiments to obtain the corresponding results \( y = \{ y^{(1)}, y^{(2)}, \ldots, y^{(m)} \} = f(X) \).

Several metrics exists to measure the goodness of a sampling plan. In this section we will focus on a widely used scalar criterion proposed by Morris et al.[98].

For an existing sampling plan \( X \) comprised of \( m \ n\)-dimensional samples, \( \{x^{(i)}\}_{i=1, \ldots, n} \).
we define the pairwise distance between two samples as the $l$-norm of their difference:

$$d(x^{(i)}, x^{(j)}) = \left( \sum_{d=1}^{n} |x_d^{(i)} - x_d^{(j)}|^l \right)^{\frac{1}{l}}$$  \hspace{2cm} (2.2)$$

where $x_d^{(i)}$ is the $d$-th element of the $i$-the vector in $X$. Note when $l = 1$ we have a *Manhattan distance*, and when $l = 2$ a *Euclidean distance* is defined.

Given this definition of $d(x^{(i)}, x^{(j)})$, we sort the the $k = \binom{m}{2}$ pairwise distances in ascending order to form a distance vector $\{d_1, ..., d_k\}$, where in [98], and many other literatures, these are named *inter-site distances*. Then let $J_i$ be the number of sample pairs that can be separated by $d_i$, we have another vector $(J_1, ..., J_k)$ called the *index vector*.

The DOEs are thus ranked with the following scalar metric where a lower value of $\Phi_q$ indicated a better design:

$$\Phi_q(X) = \left( \sum_{i=1}^{k} J_i d_i^{-q} \right)^{\frac{1}{q}}$$  \hspace{2cm} (2.3)$$

The issue of choosing an adequate $q$ is still open in this definition. When $q$ is large enough, the best plan found using Equation (2.3) will coincide with the well-know *maximin* criteria proposed by Johnson *et al.*[61], but such $\Phi_q$’s are difficult to optimize. Instead, $q$ can be 20 to 100, depending on the complexity of the experiment design problem[33].

With this scalar metric, the pursuit of a high quality DOE is then transferred to an optimization task that can be handled with global optimization algorithms such as simulated annealing[98] or genetic algorithms[20].
2.2.1 Latin Hypercube Sampling

Due to the limitation of computation, it is desirable to have a small sample number while ensuring uniform coverage of the design space. A brute force way is to construct a full factorial stratification, but it is too expensive to be feasible, hence fractional stratification designs are more practical. In this section, we introduce the Latin Hypercube Sampling (LHS), a widely used technique for constructing sampling plans.

The primary goal of stratified sampling is to ensure the uniform distribution of projections on all variable axes. To generate a sample plan with size $m$ and dimension $n$, LHS evenly divides each dimension into $m$ bins, and place samples in the bins such that each occupied bin can escape the design space along any dimension without encountering other occupied bins.

For illustration, Figure 2.2 shows a 2-D LHS designs on the design space $[0, 1]^2$ using 10 points.

2.3 Model Construction

Based on the sampling plan introduced in the previous section, we can obtain the corresponding evaluations from simulations/experiments. In this part, we will conduct an overview of the predictive model types used in this dissertation, while the details on each model type are distributed throughout the following chapters.

Broadly speaking, there are two categories of models: parametric models and nonparametric models.

Parametric models assume there exists a finite set of parameters $w$ that fully characterize
the data under investigation. That is to say, once the parameter \( w \) is learned, all future predictions on the input variable \( x \) can be made without referring to the training data set \( D: P(x|w, D) = P(x|w) \). Therefore, the complexity of parametric models is bounded by the parameter number, even if the data set is unbounded, but in return, it has a constant time complexity regardless of how large the training set is.

On the other hand, nonparametric models assume that the intrinsic data distribution cannot be defined using such a finite parameter set, the "parameter set" should have infinite dimensions, another way of interpreting it is that it is a set of "functions". Nonparametric models have better flexibility than parametric ones, but their speed and scalability is harmed when dealing with big data set.

In this section, we will discuss the simplest yet most fundamental parametric model
that can be extended to various more complicated forms - the linear regression model, and leave the discussion of nonparametric model including Gaussian processes and Student-\(t\) processes in Chapter 4.

2.3.1 Linear Regression

Suppose the input dimension is \(n\) and we have \(m\) training samples, given the sample set \((X, y) = (x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), ..., (x^{(m)}, y^{(m)})\), in which \(X \in \mathbb{R}^{(m \times n)}, x \in \mathbb{R}^n, y \in \mathbb{R}^{(1 \times n)}\), and \(y \in \mathbb{R}\), our goal is to find a mapping function \(\hat{f} : \mathbb{R}^d \rightarrow \mathbb{R}\) that represents the input-output relationships and can be used to predict the results of unseen samples.

As a first step, we consider \(\hat{f}\) to be a linear combination of the inputs:

\[
\hat{f}(x, w) = w_0 x_0 + w_1 x_1 + w_2 x_2 + ... \tag{2.4}
\]

where the model parameters \(w = \{w_0, w_1, w_2, \ldots\}\). In particular, \(w_0\) is called the intercept term and \(w_1, w_2, \ldots\) are the weights of corresponding input variables \((x_1, x_2, \ldots\)\). We conventionally set a dummy variable \(x_0 = 1\) for the convenience of compact matrix multiplication notation, therefore, Equation (2.4) is rewritten as:

\[
\hat{f}(x, w) = w^T x. \tag{2.5}
\]

Intuitively, we can use the differences between model predictions and the true observed results to quantify the quality of the linear model. The loss function \(L(w)\) is:

\[
L(w) = \frac{1}{2} \sum_{i=1}^{m} \left( \hat{f}(x^{(i)}, w) - y^{(i)} \right)^2. \tag{2.6}
\]
Note, the coefficient $\frac{1}{2}$ at the beginning is used for simplifying notations, as we shall see later. The "learning" process is essentially finding the $w$ which minimizes $L(w)$, this is commonly known as least square estimation.

There are two ways of finding the $w$ that has the minimum loss: one can either derive the normal equations, when the closed form equation is available, or use optimization algorithms such as gradient descent. Here we show the former approach.

First, rewrite Equation (2.6) in the matrix form below:

$$L(w) = \frac{1}{2}(Xw - y)^T(Xw - y). \quad (2.7)$$

To find the derivatives of $L(w)$ with respect to $w$, we have:

$$\nabla_w L(w) = \nabla_w \left( \frac{1}{2}(Xw - y)^T(Xw - y) \right)$$
$$= \frac{1}{2} \nabla_w \left( w^TX^TXw - 2y^TXw + y^Ty \right)$$
$$= X^TXw - X^Ty. \quad (2.8)$$

The second step holds because $y^TXw$ is a scalar.

As the Hessian $\nabla^2_w L(w) = X^TX$ is positive semi-definite, the $w$ that minimizes $L(w)$ is indeed the $w$ that makes $\nabla_w L(w) = 0$. Therefore:

$$w = (X^TX)^{-1}X^Ty. \quad (2.9)$$

Equation (2.9) provides a direct solution of $w$ from the training set. However, for it to hold, $(X^TX)^{-1}$ must exist, which is to say $X^TX$ cannot be singular, this condition is often
violated in real applications, and we will further address it in the next part.

2.3.2 Probabilistic View of Linear Regression

In the previous section, we formulated the linear regression problem using least square approach. We now switch to the probabilistic formulation.

Assume the output is corrupted by random errors that contain random noise or uncaptured effects, denoted as $\varepsilon$, we can update Equation (2.5) as:

$$ y^{(i)} = w^T x^{(i)} + \varepsilon^{(i)}. \quad (2.10) $$

In canonical statistical learning, aside from the assumption that $y$ has linear relationship with $x$, four more assumptions about $\{\varepsilon^{(i)}\}$ need to be satisfied, namely:

1. **Independence**: $\{\varepsilon^{(i)}\}$ are independent
2. **Homoscedasticity**: $\{\varepsilon^{(i)}\}$ have constant variance
3. **Normality**: $\{\varepsilon^{(i)}\}$ are Normally distributed with a zero mean

It is reasonable to model these errors as identical and independently distributed (i.i.d) Gaussian random variables $\varepsilon \sim \mathcal{N}(0, \sigma^2)$, where the noise level is centered at 0 with a constant variance $\sigma^2$. 

35
Therefore:

\[ \epsilon^{(i)} = (y^{(i)} - w^T x^{(i)}) \sim \mathcal{N}(0, \sigma^2) \]  

\[ p(y^{(i)}|x^{(i)}; w) = \frac{1}{\sqrt{2\pi}\sigma} \exp \left( -\frac{(y^{(i)} - w^T x^{(i)})^2}{2\sigma^2} \right). \]

Here, the notation \( p(y^{(i)}|x^{(i)}; w) \) means the probability is not conditioned on \( w \), as they are not random variables.

Equation (2.12) gives the probability that \( y^{(i)} \) happens when the the input \( x^{(i)} \) is given to the model which is parameterized by \( w \), this is the same as the probability that the model correctly predicts the result of \( x^{(i)} \). As the error terms \( \epsilon \)'s are i.i.d. distributed, the overall probability of a correct model is the probability product:

\[ L(w) = \prod_{i=1}^{m} p(y^{(i)}|x^{(i)}; w) \]

\[ = \prod_{i=1}^{m} \frac{1}{\sqrt{2\pi}\sigma} \exp \left( -\frac{(y^{(i)} - w^T x^{(i)})^2}{2\sigma^2} \right) \]

Equation (2.13) is commonly known as the likelihood of the model, and the principle of maximum likelihood gives a way of choosing the best model by maximizing it. However, in practice, maximizing (2.13) directly will complicate derivation and give rise to arithmetic underflow in floating number calculation, we instead use the strictly increasing natural logarithm function to transform \( L(w) \) into the log likelihood \( l(w) \):
\[ l(w) = \log L(w) \]
\[ = \log \prod_{i=1}^{m} \left( \frac{1}{\sqrt{2\pi\sigma}} \exp \left( -\frac{(y^{(i)} - w^T x^{(i)})^2}{2\sigma^2} \right) \right) \]
\[ = \sum_{i=1}^{m} \log \frac{1}{\sqrt{2\pi\sigma}} \exp \left( -\frac{(y^{(i)} - w^T x^{(i)})^2}{2\sigma^2} \right) \]
\[ = m \log \frac{1}{\sqrt{2\pi\sigma}} - \frac{1}{2\sigma^2} \sum_{i=1}^{m} (y^{(i)} - w^T x^{(i)})^2 \tag{2.14} \]

Apparently, maximizing (2.14) is equivalent to minimizing \( \frac{1}{2} \sum_{i=1}^{m} (y^{(i)} - w^T x^{(i)})^2 \), and hence corresponds to the least-square loss function defined in (2.9). The \( w \) obtained through maximizing \( l(w) \) is called the maximum likelihood estimate (MLE):

\[ w_{MLE} = (X^T X)^{-1} X^T y \tag{2.15} \]

Theoretically, the MLE estimate minimizes the Kullback-Leibler divergence (KL divergence), which can be thought as the "distance" from the (true) underlying distribution to the model distribution. The proof is beyond the scope of this dissertation and we direct the interested readers to [10] for a thorough treatment.

### 2.3.3 Nonlinear Features and Basis Functions

In the previous section, we have discussed how to use linear regression to construct a predictor based on the existing data. One of the biggest limitation of the simple linear regression model is that it assumes the output to be a linear combination of the inputs, and the model becomes extremely inaccurate when nonlinear relationship prevails. In this
section, we will start with polynomial linear regression models to discuss the methods for handling nonlinearity with polynomial basis functions and then introduce the commonly used radial basis function (RBF) kernel.

2.3.3.1 Polynomial Regression Model

In Figure 2.3, we show the limitation of using simple linear regression model when dealing with nonlinearity. A natural extension to linear regression model is to use nonlinear combinations of the features. For instance, for a single variable (e.g. \( x \)), we can include higher-order terms such as \( x^2, x^3, \text{etc.} \) to introduce "more" features into the linear model. Figure 2.4 demonstrates the improved results by incorporating 2nd and 3rd order terms, respectively.

Including higher order terms in the inputs is known as feature mapping, by which the original \((n\text{-dimensional})\) inputs are mapped to a new \((k\text{-dimensional})\) feature space with the mapping function \( \phi(x) : \mathbb{R}^n \rightarrow \mathbb{R}^k \), this mapping function is also called the basis function. For instance, we can express the 3rd order case as:

\[
\phi(x) = \begin{bmatrix} x \\ x^2 \\ x^3 \end{bmatrix}
\]

Exchanging the \( x \) in basic linear regression with the basis function \( \phi(x) \), the polynomial regression is then:

\[
\hat{f}(\mathbf{x}, \mathbf{w}) = \mathbf{w}^T \phi(\mathbf{x})
\]
If we let

\[ \Phi = \begin{bmatrix} -\phi_1(x) \\ -\phi_2(x) \\ \vdots \\ -\phi_k(x) \end{bmatrix} \]

Following the same procedure of deriving the normal equation, one can also obtain the least-square solution for weights \( w \) with respect to the basis function:

\[ w = (\Phi^T \Phi)^{-1} \Phi^T y \]
This is indeed the result of substituting the $X$ in Equation (2.9) / (2.15) with $\Phi$.

### 2.3.3.2 Radial Basis Functions

There are many other basis functions besides polynomial, we will follow on by introducing one of the most popular one called the *radial basis functions*.

The radial basis function takes the following form:

$$
\phi(x) = \exp\left(-\frac{(x-c)^2}{\sigma^2}\right) 
$$

(2.20)

where $c$ is called the basis function center, and $\phi(x)$ is large when $x$ is close to the center,
while vanishes when $x$ moves away from the center. The parameter $\sigma$ controls the decay rate of $\phi(x)$ as $x$ moves away: a smaller $\sigma$ will cause a faster decay.

Suppose we have $k$ centers, the feature space is hence $k$-dimensional, and the approximation function is:

$$f^*(x, w) = w^T \phi(x) = \sum_{i=1}^{k} \exp\left(-\frac{(x - c(i))^2}{\sigma^2}\right) w_i$$ \hspace{1cm} (2.21)

One can interpret this as a weighted sum of "bell-shaped" curves centered at $c$ which can be randomly picked or simple use each sample as a center, but this will potentially introduce the risk of overfitting. Besides $w$, RBF regression adds one more parameter $\sigma$, a standard approach for picking the value of $\sigma$ is through cross validation. Both overfitting and cross validation will be covered later in this chapter.

The RBF model demonstration for the same 1-D dataset is shown in Figure 2.5.

### 2.3.4 Regularization

Recall that we mentioned the potential problem with the normal equation (2.9) in the linear regression section, where $(X^T X)^{-1}$ may not exist due to the poorly conditioned $(X^T X)$ matrix. For example, when $m = 20$ and $n = 2000$, $(X^T X)$ will seldom be full-rank, and thus invertible. One way to handle this singular matrix issue is through adding small elements along the matrix diagonal to "stabilize" it.

Another way to look at regularization is that it is trying to suppress the model complexity. The famous Occam's Razor summarizes this situation as: "If you have two equally likely solutions to a problem, choose the simplest." Hence, if two models have competing performances in fitting the training data, the simpler model should always be preferred.
In the beginning of this chapter, we introduced "Tikhonov regularization problem" (Equation (2.1)) that shows a compact representation of this simple yet effective rule, but the $\hat{f}(x)$ term is inconvenient to evaluate. Instead, people usually trade-off the model accuracy and complexity by adding a regularization term in the loss function $L(w)$ (Equation 2.6 and 2.7). The regularized least square loss function then becomes:

$$L(w) = \frac{1}{2} \sum_{i=1}^{m} \left( \hat{f}(x^{(i)}, w) - y^{(i)} \right)^2 + \frac{\lambda}{2} \sum_{i=1}^{n} |w_i|^q$$  \hspace{1cm} (2.22)

Or equivalently, using the matrix notation:

$$L(w) = \frac{1}{2} (Xw - y)^T (Xw - y) + \frac{\lambda}{2} ||w||_q^q$$  \hspace{1cm} (2.23)
Literatures give different names to the regularization methods when \( q \) varies. For example, when \( q = 1 \) (L1 norm) the model is called a \textit{lasso} regression, and when \( q = 2 \) (L2 norm), it is called \textit{Ridge} regression\cite{10}. \( q = 2 \) is commonly used to combat over-fitting, while \( q = 1 \) is often adopted when dealing with feature selections, as it tends to shrink unimportant parameters towards 0\cite{36}.

We consider the Ridge regression \((q = 2)\) in our following discussion:

\[
\nabla_w L(w) = \nabla_w \left( \frac{1}{2} (Xw - y)^T (Xw - y) + \frac{\lambda}{2} w^T w \right) \\
= \frac{1}{2} \nabla_w (w^T X^T Xw - 2y^T Xw + y^T y + \lambda w^T w) \\
= X^T Xw - X^T y + \lambda w 
\]

(2.24)

Let \( \nabla_w L(w) = 0 \), we have:

\[
w = (X^T X + \lambda I)^{-1} X^T y 
\]

(2.25)

where \( I \) is a \((n \times n)\) diagonal matrix.

Therefore, introducing the regularization term results in adding the regularization parameter \( \lambda \) along the diagonal of \( X^T X \). This is indeed where the name "ridge" stems from, and it matches with our initial intent of adding small element \((\lambda)\) along the diagonal to make it full-rank.

Similarly, for linear regression using basis function, Equation (2.19) becomes:

\[
w = (\Phi^T \Phi + \lambda I)^{-1} \Phi^T y 
\]

(2.26)
2.3.5 Overfitting and Underfitting

For surrogate models to be useful, they need to both fit the training set well and have high generalization capability to give accurate predictions for unseen data. If the model is too simple, the inability to fit the training set incurs, while on the other hand, by increasing the model complexity, such as the polynomial order or basis function quantity, the model can eventually pass through all the training data perfectly, but this is rarely useful for predicting unobserved cases. The prior issue is called underfitting and the latter overfitting.

Using the same example as before, we show the plot of under/overfitting and the preferable model with the scikit-learn Python package (Figure 2.6):

![Figure 2.6 Under-fitting, desirable model, and over-fitting. Created with the scikit-learn package](image)

As seen in the plots, both underfitting and overfitting generate undesirable results. When the model complexity is low, we say it has high bias and underfitting follows; but when it is too complex, it has high variance that leads to overfitting. In the linear regression
case, the weight number and absolute value represents the model complexity. For example, in the 1D demo, an \( n \)-degree model has \((n + 1)\) free parameters, including the interception term.

As mentioned above, the second purpose of regularization is to control model complexity by including the parameters norms in the lost function. We now show the effects before and after adding regularization term with \(\lambda = 1\) and \(\lambda = 1e100\) for a 14-degree polynomial model in Figure 2.7. A small regularization parameter will significantly improve the model performance. But if the regularization parameter is too large, all \(w\)’s vanish, leaving us with a (almost) straight line. It remains an open problem for now on how to select the best regularization parameter, and we will deal with this in the next section using cross validation.

**Figure 2.7** Regularization for over-fitting. Created with the scikit-learn package
2.4 Model Selection and Cross Validation

Although regularization provides an approach for controlling model complexity and preventing matrix instability, it also adds another parameter - the regularization strength $\lambda$. The problem of choosing an appropriate $\lambda$ is a part of the model selection process and cross validation is a typical approach for this purpose.

The basic idea of (simple / hold-out) cross validation is to randomly split the training data ($D = \{X, y\}$) into two sets - the training set ($D_{\text{train}}$) and the cross validation set ($D_{CV}$). Each candidate model is first trained on the training set, and their respective performances are then validated using the cross validation set. In the end, the model with the lowest cross validation error is selected as the best model.

Seemingly intuitive, there are two hidden problems with the simple hold-out cross validation algorithm. Firstly, it wastes a subsection of the training data. This may not be a huge problem when abundant data is available, but under data scarcity, which applies especially to the surrogate model-based engineering design context, hold-out cross validation will further exacerbate this situation. On the other hand, suppose we have a large quantity of candidate model at hand, evaluating every model with the cross validation set and pick the top-performing one is equivalent to training the model with $D_{CV}$, which will consequently lead to overfitting the cross validation set.

To alleviate the problems mentioned above, one prefers to use the $k$-fold cross validation technique. We summarize the main steps below[104]:

1. Randomly split the training data $D$ equally into $k$ disjoint sets: $\{D_i\}_{i=1:k}$ (therefore, $|D_i| = m/k$)
2. For each model $M_i$, train the model sequentially on $\{D \setminus D_j\} \forall j = 1, \ldots, k$ and test the model on $D_j$ to get the error $\hat{\epsilon}_{ij}$. The model accuracy for $M_i$ is then $\hat{\epsilon}_i = \sum_{j=1}^{k} (\hat{\epsilon}_{ij})$

3. Pick the model $M_i$ which has the lowest $\hat{\epsilon}_i$ and retrain it on $D$, this is the final model.

While $k = 10$ is a common choice, under extreme data scarcity, we will use $k = m$ that results in holding only one sample out for each fold. This method is known as leave-one-out cross validation.

2.5 Adaptive Sampling

Up until now, our discussions on model training and selection are based on a pre-selected training data set. However, in practical engineering design modeling, this "one-shot" approach is usually not enough to deliver a globally accurate surrogate model. For instance, even though the DoE step generates a sampling plan that uniformly covers the design space, local nonlinearity or discontinuity may not be well captured. To overcome this issue, we will use sequential design approach involving adaptive sampling, also called the active learning technique[21].

Adaptive sampling is an iterative process of acquiring new samples by analyzing the current model, and the new samples are located in areas that are harder to be approximated. The resulting data distribution is efficient, as it ensures the design space coverage while paying particular attention to the key regions.

There are typically two classes of methods, namely exploration-based and exploitation-based methods. The former one tries to select samples in such a way that the input domain is covered uniformly, examples include [111]. The latter class tries to pick new samples
where the model estimation error is high, for example [126]. Both methods have their advantages and drawbacks. For instance, exploration-based method tends to overlook the difficult area (actually, the initial DoE is an example of the one-shot exploration-based method), while exploitation-based method puts too much emphasis on the nonlinear or discontinuous regions, leaving other regions unexplored. In this work, we advocate the strategy called the LOLA-Voronoi criteria[21] which takes both scenarios into consideration. The general idea is illustrated here, and we redirect interested readers to [21] for a more rigorous discussion.

In LOLA-Voronoi method, there are two components: the LOcal Linear Approximation (LOLA) and the Voronoi tessellation. The LOLA components base itself on the assumption that samples should be proportional to the local nonlinearity of the function. For instance, a linear function requires fewer samples compare to a sinusoid function. The LOLA criteria thus estimate local nonlinearity by computing the empirical gradient of a point using the points in its vicinity. On the other hand, in the Voronoi part, an approximation of the Voronoi tessellation is constructed using the current dataset to estimate the sample density. We show an example of 2-D Voronoi tessellation in Figure 2.8. A larger Voronoi cell size indicates a lower sample density, thus needs to add more samples. Combining both components, the LOLA-Voronoi method iteratively trades off exploitation using LOLA and exploration using Voronoi to ensure the most efficient sample distribution.

2.6 Handling High Dimensionality with HDMR

A fundamental limitation of surrogate modeling arises with the increasing problem dimensions: as the number of sampling points needed to give reasonably uniform coverage of the
design space rises exponentially, this is known as the *curse of dimensionality*\cite{33}.

To alleviate this situation, widely accepted practices are either limiting the range of the variables so that the response surface is sufficiently smooth to be approached with sparse data or limiting the number of input variables to exclude the ones with less influence on the overall problem. Another way of reducing the dimensionality is to use principal component analysis (PCA) based surrogate modeling approach, as in\cite{138}. The idea is to create a smaller set of features that are orthogonal to each other by combining the original features of the problem. These techniques are useful as preparatory work to handle model dimensionality, however, if all the model features have comparable contributions within the problem, or when the features are already statistically independent, the alleviation would not be significant enough.

Here, we advocate another approach called high dimensional model representation
(HDMR) to efficiently handle the curse of dimensionality. First developed by Sobol[123], the HDMR method is based on the divide-and-conquer philosophy that transforms a high-dimensional problem into a set of lower dimensional expressions. A general HDMR structure for a -dimensional problem is shown in Equation (2.27):

\[
f(x) = f_0 + \sum_{i=1}^{d} f_i(x_i) + \sum_{1 \leq i < j \leq d} f_{ij}(x_i, x_j) + \ldots + \sum_{1 \leq i_1 < i_2 < \ldots < i_l \leq d} f_{i_1i_2\ldots i_l}(x_{i_1}, x_{i_2}, \ldots, x_{i_l})
\] (2.27)

The component \(f_0\) is a constant number that represents the 0-th order (\(l = 0\)) effect on \(f(x)\); the component function \(f_i(x_i)\) reflects the independent effect of each input variable, which is called the 1st order (\(l = 1\)) effect; 2nd order (\(l = 2\)) effect component \(f_{ij}\) refers to the interactive effect between input \(i\) and input \(j\) on the final output, and this definition holds for all the \(n\)-th order effect component functions. Genyuan et al. [59] shows that order \(l \leq 2\) can often provide a satisfactory description for many high dimensional systems.

Several options exist for determining the component functions, such as RS-HDMR (random sample HDMR) and Cut-HDMR[82]. RS-HDMR is mainly used for generating HDMR from lab or field experiments, whose samples are pre-designated and cannot be chosen at will during the process of modeling. While Cut-HDMR involves only simple arithmetic computation and presents the least costly model with similar accuracy as other HDMR types when sampling for outputs at chosen points is possible. Therefore we use Cut-HDMR to determine the component functions in our modeling flow.

Cut-HDMR expresses \(f(x)\) as a superposition of its values on lines, planes and hyperplanes (or cuts) passing through a "cut" center which is essentially a point in the input variable space. The component functions of the Cut-HDMR with respect to the reference
point $x_0$ are listed as follows[82]:

$$f_0 = f(x_0)$$  \hspace{1cm} (2.28)

$$f_i(x_i) = f(x_i, x_0^i) - f_0$$  \hspace{1cm} (2.29)

$$f_{ij}(x_i, x_j) = f(x_i, x_j, x_0^{ij}) - f_i(x_i) - f_j(x_j) - f_0$$  \hspace{1cm} (2.30)

... 

Where $x_{i0}$ means all the points are fixed at the cut-center except for the ith input which is a variable in this expression, this definition also holds for $x_{0}^{ij}$ and so on.

The HDMR approach provides tremendous benefit on computing power as it converts the usage of samples from exponential to polynomial dependence on the problem dimensionality. As shown in [114], the required number of samples to model a $d$ dimensional problem is reduced from (2.31) to (2.32):

$$N_{direct} = s^d$$  \hspace{1cm} (2.31)

$$N_{HDMR} = \sum_{i=0}^{l} \frac{d!}{(d-i)!i!}(s-1)^i$$  \hspace{1cm} (2.32)

Here $s$ denotes the number of samples along each dimension. Consider the highest level which takes the dominant number of samples, the cost is reduced to $\frac{(d,s-1)!}{(l-1)!}$. Assuming $l << d$, the exponentially increasing difficulty $s^d$ is then transformed into a polynomial complexity of $s^l$. 

51
In this chapter, we will demonstrate the application of machine learning techniques in three circuit and system design examples, including a Voltage-Controlled Delay Line (VCDL), a phase rotator, and the place and route configuration of a CORTEXM0 microprocessor.
3.1 Case Study 1: VCDL Model

As the deep sub-micron circuits are more and more prevalent in radio-frequency integrated circuits (RFIC), reliability degradation due to aging effects, such as the hot carrier injection (HCI) and negative bias temperature instability (NBTI), have raised broad interests. One way to tackle this problem is to develop healing algorithms that can calibrate the circuit automatically[134]. However, as the development requires hundreds or even thousands of function evaluations, it is practically infeasible to rely on the time-consuming SPICE level aging simulations. Due to their appealing characteristics of speed and accuracy, surrogate models will effectively solve this bottleneck.

In this examples, we show how surrogate models, a type of supervised machine learning predictive models, assist in the design exploration and calibration algorithm development using a Voltage-Controlled Delay Line (VCDL). We first briefly discuss aging-induced circuit degradation, with a focus on the Negative Bias Temperature Instability (NBTI) effect which is more prominent in the digital circuits, and then turn to the behavioral modeling of an NBTI testing infrastructure.

3.1.1 Aging Degradation and the NBTI effect

Negative bias temperature instability (NBTI) is one of the most concerned reliability issues as technology continues to scale down below 130nm. The mechanism of NBTI is such, when a negative bias is applied to the transistor across the gate and source at elevated temperature (80C -150C), the electrical field will break the Si-H bonds at the channel-oxide interface[74], resulting in dangling Si bonds as well as hydrogen (H or H2) defusing back to
the gate insulator through the dielectric. Missing hydrogen creates traps at the interface, according to the R-D model, device characteristics shift will be caused by these traps, such as threshold voltage ($V_{th}$) increase and a drop in the effective channel carrier mobility $\mu_{eff}$ [2].

The worst case scenario of NBTI happens in PMOSFETs when the gate is grounded (fully-on, strongest inversion), since the holes in the PMOS inversion layer are known to interact more with the oxide states than in NMOS[140]. In this case, $V_{DS}$ is very low and reliability issues caused by effects like hot carrier injection (HCI) that requires high drain currents can be neglected. Since strong inversion is often observed in digital FETs, NBTI effect manifest itself a significant impact on the long time performance of digital logic circuits. For instance, researchers have shown that it will result in degradation in the static noise margin of SRAMs[76], and also decrease the maximum operating frequency of digital circuits[75, 76].

NBTI is known to recover once the stress is removed partially[4], hence the theories derived from static NBTI testing, assuming the device is statically negatively-biased, will tend to overestimate the degradations in digital logic circuits. Instead, investigation of dynamic NBTI effect is essential to avoid the over-design of digital circuits. Previous works such as[25, 67, 74, 131] presents methods focusing on the dynamic NBTI effect monitoring and predictive modeling techniques, which provide guidance on NBTI-prone circuits design. However, to obtain reliable operating over an extended time of use, we need to have in-situ investigation as well as calibration of the stressed devices.

In this section, we discuss the behavioral modeling of an automated closed-loop test infrastructure that can collect real-time NBTI degradation data and heal the NBTI effect by
applying calibration algorithms. The cores of this infrastructure are two Voltage-Controlled Delay Lines (VCDLs).

A VCDL is a chain of inverters that can be powered by different supply voltages to achieve various delays. Complex digital CODECs require the accurate delays of a VCDL for communication[136], it is also an essential component of the Phase Lock Loop (PLL) and is very sensitive to changes in the transistor threshold voltage ($V_{th}$). Figure 3.1 shows the block diagram of the NBTI test infrastructure, and the blue-shaded part of the diagram are two 45nm VCDLs, namely the aging delay line and the reference delay line. The power supply for the aging delay line is controlled by an 8-bit DAC, while the phase detector measures the phase difference between these two delay lines.

![Figure 3.1 Block diagram of the VCDL test infrastructure](image)

Figure 3.1 Block diagram of the VCDL test infrastructure
Due to NBTI effects, the aging delay line will become slower over time, which results in a phase shift at the output of the aging line. If the output of the reference delay line stays the same, the output voltage of the phase detector (CompOut) will change. From the CompOut values, we can both determine how much aging effect is accumulated as well as how to tune the DAC to heal the aging effect to make the two VCDLs maintain a target phase difference.

From a modeling perspective, the VCDL problem is 2-dimensional, as summarized in Table 3.1.

In this example, we use the SUMO Toolbox for the surrogate modeling task[44]. The surrogate modeling goal is to arrive at a simplified mathematical description of the system by evaluating a relatively small number of points in the state space of the system. The response of the system at these points is compared to the response of detailed simulations using the best available model. The results of the comparisons are used to iteratively adjust parameters in the surrogate model to improve accuracy until the accuracy level meets a predetermined goal. The model accuracy is measured in root relative squared error (RRSE), as defined in Equation (3.60). This reflects how well the model is fitted over the mean value, where, and indicates the actual, modeled and mean actual value, respectively.

\[
RRSE = \sqrt{\frac{\sum_{i=1}^{n} (y_i - \hat{y}_i)^2}{\sum_{i=1}^{n} (y_i - \bar{y})^2}} \tag{3.1}
\]
As we mentioned, the SPICE level aging simulation is extremely expensive, to speed up the surrogate model training process, we implemented a multi-server simulation flow, shown in Figure 3.2. The flow is well-suited for parallelization, which greatly reduces the total modeling time. Moreover, if the desired accuracy is not reached, we use the adaptive sampling feature to analyze the data from previous iterations and select data in the region that are not well captured, hence a more efficient distribution of samples can be generated.

**Figure 3.2** The parallel simulation flow
For the model type, we select RBF model due to its capability of handling nonlinearity. The model takes 392 samples, and the final RRSE goal is set to be 10%. Shown in Figure 3.3 is the modeling result for this VCDL problem using the parallel framework discussed in the previous session. In this plot, red color means higher CompOut value while blue means lower. With this plot, we can see what DAC settings we need change to in order to compensate for the aging effect and maintain the same level of raw phase detector output value.

Figure 3.3 Contour Plot of the VCDL Model

3.2 Case Study 2: Phase Rotator Model

The phase rotator problem is a more complicated 45nm SOI test case for reliability calibration.
Table 3.2 Phase Rotator Inputs and Outputs Summary

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Outputs</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Knobs</strong></td>
<td><strong>Ranges</strong></td>
</tr>
<tr>
<td>Age</td>
<td>0 - 10 year</td>
</tr>
<tr>
<td>PRI</td>
<td>0 - 63</td>
</tr>
<tr>
<td>PRQ</td>
<td>0 - 63</td>
</tr>
<tr>
<td>I/Q Drain</td>
<td>0.1 - 0.9 V</td>
</tr>
<tr>
<td>I/Q NGate</td>
<td>0.1 - 0.9 V</td>
</tr>
<tr>
<td>I/Q PGate</td>
<td>0.1 - 0.9 V</td>
</tr>
</tbody>
</table>

We show a conceptual block diagram in Figure 3.4. The phase rotator is based on the design in [130] and contains two components: a baseline component and a self-healing one. Similar to the VCDL example, the baseline design is to achieve desired performance under typical operating conditions, while the self-healing portion is to cope with performance degradations.

Using the 3 GHz in-phase (I) and quadrature (Q) local oscillator (LO) inputs, the basic operating principle of the phase rotator is to generate a target phase relative to the in-phase (I) LO input at a user specified target gain level. The baseline turning knobs adjust the gains of the variable gain amplifiers (VGAs) until the phase rotator output is within some specified error tolerance relative to the chosen phase and gain targets. To compensate for performance degradation effects at the 45nm node such as the hot-carrier injection (HCI) effect [53], six additional self-healing tuning knobs (each having 8-bits of resolution) were integrated into the phase rotator design. The reader is directed to [134] for a detailed discussion.

The input tuning knobs and outputs of the phase rotator model is summarized in Table 3.2.
Figure 3.4 Block diagram the phase rotator design

In Figure 3.5, we show a point-by-point simulation of the RF phase rotator's phase response. Notice that there is a sharp "cliff" in the response surface due to the wrapping from 0 degree to 360 degree. In developing the calibration algorithm, we would like the algorithm to be resilient enough to handle this highly nonlinear search space.

This RF phase rotator test case has a response space containing $2^{60}$ total points (i.e., there are totally $2^{60}$ distinct digital tuning knob settings), and performing $2^{60}$ circuit simulations for calibration performance validation purposes is extremely prohibitive. In order to enable a detailed calibration algorithm performance and reliability analysis for the phase rotator test case, the gain and phase responses were modeled using our proposed supervised machine learning framework with polynomial models.
However, direct modeling of this high dimensional nonlinear response surface lead to an over-simplification of the problem, as shown in Figure 3.6. The cliff trait is smoothed out due to non-sufficient samples under the curse of high dimensionality.

Using the HDMR approximation given in (2.27), which leads to 29 component functions in total for this problem, we are able to reconstruct the desire phase wrapping characteristics. The automatic LOLA-Voronoi enhanced HDMR flow is shown in Figure 3.7.

To obtain the 29 surrogate models, a total of 2894 circuit simulation samples were executed. In Figure 3.6 we show the modeled phase response as a function of the two baseline tuning knobs. Compared to the $2^{60}$ total number of points in the response space, the 2894 samples required for surrogate model generation is a highly-significant reduction in the computational complexity required for response generation. We should note that, as shown in Figure 3.8, there is a sharp transition from 0 degree to 360 degree, and to
adequately cope with the modeling complexity in this region, we set the RRSE target to be 0.1, or 10%. For completeness, in Figure 3.9, we also show the modeled gain response. Here, due to the "smooth" response behavior, we set the RRSE target to be 0.01, or 1%. Also note that we rotate the response surface to view the cliff from a clearer angle, but the plot is exactly correspond to the one shown in Figure 3.5.

### 3.3 Case Study 3: Model-Based Physical Design

In the previous two sections, we have discussed the applications of machine learning in building predictive models for RFICs to assist the calibration algorithm development. However, this general approach can also be extended into digital IC field and help physical
Figure 3.7 LOLA-Voronoi HDMR Modeling Flow
Figure 3.8 The Phase Rotator Phase Response Model

Figure 3.9 The Phase Rotator Magnitude Response Model
designers find an optimum configuration.

Physical design, which is also called the back-end design in the modern IC design flow, is a step of converting components (devices and interconnects) of the IC design into geometric representations of shapes which, when manufactured in the corresponding layers of materials, will ensure the required functioning of the components. Steps such as floor-planning, clock tree synthesis, and power plan are crucial to achieving an optimal performance-cost trade-off in deep sub-micron system-on-chip physical design.

The design example we use here is a CORTEX system-on-chip (SoC) platform (Figure 3.10). Our goal is to successfully place and route this design using a minimum number of layers and highest density while keeping it fully functional using lowest possible clock period with no setup and hold violations. The place and route tool in our test case is the Encounter by Cadence\textsuperscript{TM}. The complex interactions of layer number, clock period and density along with other configurations in Encounter make the system a black-box. Hence designers have to spend an enormous amount of time exploring the design space to achieve better performances.

Consider the advantages of machine learning models in handling black box systems, we can alleviate human effort involved in the design space exploration by running the automatic flow to sample and model the design. The model parameters are summarized in Table 3.3.

We first execute 36 physical design runs as our initial design for preliminary design parameter and design space exploration. Figure 3.11 - 3.13 shows the relationships from maxSkew and sinkMaxTran to Congestion, Setup slack, and Hold slack, respectively. Apparently, maxSkew and sinkMaxTran only have a significant impact on the hold time, while
Figure 3.10 The Macro-Structure of the CORTEX SoC Platform

Table 3.3 CORTEX Physical Design Parameter Summary

<table>
<thead>
<tr>
<th>Input</th>
<th>Knobs</th>
<th>Ranges</th>
<th>Outputs</th>
</tr>
</thead>
<tbody>
<tr>
<td>Clock Period</td>
<td></td>
<td>10 - 40 ns</td>
<td>Congestion</td>
</tr>
<tr>
<td>Density</td>
<td></td>
<td>0.2 - 0.9</td>
<td>Hold time slack</td>
</tr>
<tr>
<td>Number of Layers</td>
<td></td>
<td>3 - 10</td>
<td></td>
</tr>
<tr>
<td>maxSkew</td>
<td></td>
<td>100 - 300</td>
<td>Setup time slack</td>
</tr>
<tr>
<td>sinkMaxTran</td>
<td></td>
<td>100 - 400</td>
<td></td>
</tr>
</tbody>
</table>
the effect on congestion and setup time is trivial.

Therefore, in our following detailed modeling stage, we will only consider the three inputs, namely Clock Period, Density and Layers, and sinkMaxTran and maxSkew will be used to increase the hold time slack in the prevention of hold-time violations.

Moreover, we don’t need to perform the entire final route stage to determine if there would be violations. Instead, this can be implied from the faster trail route stage. According to the data we accumulated in the initial design, when the congestion at trial route stage is less or around 0.1%, for both vertical and horizontal direction, there will typically not be DRC errors in the final route stage. Hence, we can adopt the faster trail route statistics to help us to determine beneficial information. A detailed study of the mapping from trail route stage to the final route stage is discussed in [80].

The primary objective is to minimize the clock period, so we set our goal to 10ns. The

**Figure 3.11** Trivial Effects of sinkMaxTran and maxSkew on Congestion
**Figure 3.12** Trivial Effects of sinkMaxTran and maxSkew on Setup slack

**Figure 3.13** Significant Effects of sinkMaxTran and maxSkew on Hold slack
Figure 3.14 Congestion vs. Layers (CLKper = 10, density = 0.6)

Table 3.4 Results of iteration 1

<table>
<thead>
<tr>
<th>Period</th>
<th>Density</th>
<th>Layer</th>
<th>max Skew</th>
<th>Max Tran</th>
<th>Congestion</th>
<th>Violaton</th>
<th>Hold slack</th>
<th>Setup slack</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>0.6</td>
<td>8</td>
<td>300</td>
<td>400</td>
<td>0.28H / 1.51V</td>
<td>105</td>
<td>-61.3 ps</td>
<td>6.46 ps</td>
</tr>
</tbody>
</table>

Comment: over-congested, hold time slack violated

density of standard cells should not be too low nor too high if we want to achieve a good function-cost trade-off, and initially, we set it to 0.6. The following figure (Figure 3.14) shows the Congestion vs. layers, with clockper = 10 and density = 0.6. From this plot, we see that the layer should be greater than 8. Hence, we perform our first iteration (CLKper = 10, density = 0.6, layers = 8), with results in Table 3.4.

The result from Iteration 1 indicates that the design is over-congested. But using this as a starting point, we can fine-tune the physical design parameters, as listed in Table 3.5.

It is shown that with the help of predictive models, we arrive at the final specification-
Table 3.5 Physical Design Iterations

<table>
<thead>
<tr>
<th>Iter</th>
<th>Period</th>
<th>Density</th>
<th>Layer</th>
<th>max Skew</th>
<th>sink Max Tran</th>
<th>Congestion</th>
<th>Violation</th>
<th>Hold slack (ps)</th>
<th>Setup slack (ns)</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>10</td>
<td>0.5</td>
<td>8</td>
<td>300</td>
<td>400</td>
<td>0.03H/0.39V</td>
<td>6</td>
<td>-48.7</td>
<td>6.55</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Comment: Over-congested, hold time slack violated</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>10</td>
<td>0.45</td>
<td>8</td>
<td>300</td>
<td>400</td>
<td>0.02H/0.11V</td>
<td>0</td>
<td>2.4</td>
<td>6.48</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Comment: No DRC error; hold time fixed, hold margin is low</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>10</td>
<td>0.45</td>
<td>8</td>
<td>200</td>
<td>300</td>
<td>0.02H/0.17V</td>
<td>0</td>
<td>10.5</td>
<td>6.37</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Comments: Final Design</td>
<td></td>
</tr>
</tbody>
</table>

met design with additional three iterations. The final system design details are listed in Figure 3.15, along with the layout snapshot in Figure 3.16.

### 3.4 Summary

In this chapter, we have demonstrated the applications of machine learning predictive models in handling time-consuming large-scale circuit and system designs with three practical design problems. Test case results show that model-based design can substantially reduce the simulation time and human effort in developing calibration algorithms and designing circuit systems. In the next chapter, we will introduce an automated design analysis and parameter tuning technique to minimize the human labor spent in design and reuse intellectual properties.
1. Area (Resulted from summaryReport/cortex_soc.main.html)

<table>
<thead>
<tr>
<th>Number of Standard Cells</th>
<th>39990</th>
</tr>
</thead>
<tbody>
<tr>
<td>Area ((\mu m^2))</td>
<td>Core 98109.284 (313.224*313.224)</td>
</tr>
<tr>
<td></td>
<td>Chip 54363.008 (233.158*233.158)</td>
</tr>
<tr>
<td>Cell Density</td>
<td>55.4 %</td>
</tr>
</tbody>
</table>

2. Routing (Resulted from run_route.log)

<table>
<thead>
<tr>
<th>Number of Metal Layers</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td>Total Wire Length ((\mu m))</td>
<td>347202.73</td>
</tr>
<tr>
<td>Average wire length ((\mu m))</td>
<td>16.5169</td>
</tr>
</tbody>
</table>

3. Timing and Signal Integrity

<table>
<thead>
<tr>
<th>Target Clock Rate (ns)</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>Setup Slack (ns)</td>
<td>6.3735</td>
</tr>
<tr>
<td>Hold Slack (ns)</td>
<td>10.5</td>
</tr>
<tr>
<td>Worst case Noise by Area Height</td>
<td>0.2772</td>
</tr>
<tr>
<td></td>
<td>Width</td>
</tr>
</tbody>
</table>

4. Clock

<table>
<thead>
<tr>
<th>Maximum</th>
<th>Minimum</th>
</tr>
</thead>
<tbody>
<tr>
<td>Setup Latency</td>
<td>0.1440 (Launch)</td>
</tr>
<tr>
<td>Hold Latency</td>
<td>0.1119 (Launch)</td>
</tr>
<tr>
<td>Active Transition</td>
<td>0.0580</td>
</tr>
<tr>
<td>Max. Setup Skew</td>
<td>0.0306</td>
</tr>
<tr>
<td>Max. Hold Skew</td>
<td>0.0309</td>
</tr>
<tr>
<td>Nr. of Stages</td>
<td>4 (including gates)</td>
</tr>
<tr>
<td>Nr. of Sinks</td>
<td>2053</td>
</tr>
</tbody>
</table>

5. Power and Power Integrity

<table>
<thead>
<tr>
<th>cortex_soc</th>
<th>ahb</th>
<th>mem_ctl</th>
<th>CORTEXMODS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Switch Pwr</td>
<td>8.38e-5</td>
<td>1.67e-5</td>
<td>1.39e-5</td>
</tr>
<tr>
<td>Int Pwr</td>
<td>2.12e-4</td>
<td>9.26e-7</td>
<td>1.23e-4</td>
</tr>
<tr>
<td>Leak Pwr</td>
<td>2.01e-3</td>
<td>6.61e-5</td>
<td>1.04e-3</td>
</tr>
<tr>
<td>Total Pwr</td>
<td>2.31e-3</td>
<td>9.20e-5</td>
<td>1.18e-3</td>
</tr>
<tr>
<td>Worst VDD IR Drop</td>
<td>0.950V</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Worst VSS IR Drop</td>
<td>1.373mV</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Figure 3.15** Final design details
Figure 3.16 Snapshot of the final routed design
Technology advancement requires reusing analog intellectual properties (IPs) across different nodes. In this chapter, an efficient statistical learning-based methodology is presented to aid designers in analog IP analysis, redesign, and reuse. Two test cases, including a passive balun and a power amplifier, are optimized in the original node (IBM 8XP) and reused in a newer one (IBM 9HP). Results indicate that the proposed approach automatically
delivers satisfactory designs and efficiently assists in design parameter determination and design goal trade-offs. The simulation cost scales linearly with dimensionality, and the Bayesian optimization we advocate is over three times more computationally efficient than the traditional evolutionary algorithms.

4.1 Introduction

Analog design is challenging as it requires deep design expertise and intensive human effort. Consequently, analog designs often become the bottleneck of the time-to-market[41]. Under rapid technology advancement, the well-designed analog circuit blocks, or intellectual properties (IP), need to be reused across technology nodes. However, the technology dependency of analog IP often implies a new round of design from scratch resulting in yet another a long design cycle[132].

Various CAD tools emerged to alleviate this situation, which can be broadly summarized into knowledge-based and optimization-based approaches[7]. Knowledge-based approaches require encoding expert knowledge into the design flows, though with fast execution speed, defining and reforming such plans requires tremendous efforts; on the other hand, optimization-based approaches transforms the design into an optimization problem of finding the optimal values in the design space, that can be solved automatically with optimization algorithms. The latter proves to be more favorable due to their nature of automaticity. However, optimization algorithms tend to demand a huge number of function evaluations that consumes unaffordable computational power. Insufficient design analysis that leads to over- or underestimated design objectives or defining more than enough design parameters even aggravates this situation, which potentially causes ill-formed or
even unfeasible design targets.

In the hope of solving the challenges mentioned above, we propose a computationally efficient statistical learning-based methodology that automatically performs design analysis, IP redesign and reuse which will benefit designers with automatic design knowledge extraction and fast design turnaround.

This chapter is organized as follows: in Section 4.2 we present an overview of the proposed methodology, and in Section 4.3 the design analysis techniques including sampling strategy and data manipulation follows. The theory of Bayesian optimization, which is the key to this flow, is presented in Section 4.4, and we demonstrate the efficacy of the proposed workflow with two design examples (77GHz balun and power amplifier) in Section 4.5. Finally in Section 4.6, the conclusion and contribution are discussed.

### 4.2 Overview of Proposed Approach

We propose a computationally efficient workflow based on statistical learning for design analysis and optimization. The basic idea is to identify the most influential parameters and use statistical surrogate models in place of the expensive simulator to significantly reduce the computational cost. Two key components are included: design analysis selects critical design parameters, analyze and visualize the design goal trade-offs, and automatically formulate objective function; while design optimization performs Bayesian optimization with Gaussian process model or Student-\(t\) process model for analog IP redesign and reuse. The workflow is shown in Figure 4.1.

Aiming at general applicability, this flow can be tailored based on the actual need. For design analysis, one provides a netlist, and the tool will deliver the top influential design
parameters, along with the achievable design goal distributions and correlations. These results reveal the underlying relationships and potential trade-offs either visually or statistically. For IP redesign, one can either manually define the design parameters and objective functions if the design is well understood, or let the tool automatically configure via design analysis. For reusing the IP, the tool will first validate the similarity between the old and the new designs using a small validation set, IP redesign follows if significant differences are detected; otherwise the design analysis stage is skipped, and the optimization configurations are inherited from the old design. In both cases, the tool proceeds with design optimization in the new node for reusing the IP.

4.3 Design Analysis

4.3.1 Design Parameter Screening

Design space grows exponentially with parameter count, therefore, it is desirable to identify the more important ones. Traditionally this is done with design knowledge or data-
consuming variance-based global sensitivity analysis (e.g. Sobol’s method). Instead, we use the screening method proposed by Morris[97] and modified by Campolongo[14]. This method is suitable for analog design analysis as it requires far fewer samples without strictly assuming input-output additivity or monotonicity[14].

For a $k$-dimensional design space $\Omega$, by discretizing each dimension into $p$ levels, a full-factorial grid is formed:

$$X_i \in \left\{ 0, \frac{1}{(p-1)}, \frac{2}{(p-1)}, \ldots, 1 \right\}, \forall i = 1, \ldots, k. \quad (4.1)$$

By randomly selecting a pivot point, the design parameters are varied one at a time at each level to measure the elementary effect. The elementary effect for the $i$-th input of a given starting point $X$ is therefore defined as:

$$d_i(X) = \frac{y(X_1, \ldots, X_{i-1}, X_i + \Delta, X_{i+1}, \ldots, X_k) - y(X)}{\Delta}. \quad (4.2)$$

where $\Delta \in \left\{ \frac{1}{(p-1)}, \ldots, \frac{(p-2)}{(p-1)} \right\}$.

Suppose the $i$-th elementary effect follows a finite distribution $F_i$, i.e. $d_i(X) \sim F_i$, the distribution can be estimated by constructing $r$ trajectories of $k + 1$ points in $\Omega$, each resulting in $k$ (one for each dimension) elementary effects, where $r$ can be as small as 15 to 20[14], we set $r = 50$ for a better coverage. Therefore, the total sampling cost is $50 \times (k + 1)$, depending linearly on $k$, making it very attractive for analyzing expensive designs with growing dimensionality.

Morris[97] used two measures for each design variable: $\mu$ is the estimated mean of $F_i$
and $\sigma$ the estimated standard deviation. A higher $\mu$ indicates more impact on the output while a higher $\sigma$ indicates nonlinear effect or interaction with other parameters. However, this estimate is prone to Type-II error when $F_i$ contains oscillating signs that cancel each other out due to non-monotonicity of the design, so a third measure $\mu^*$ estimating the mean of the absolute elementary effect distribution is introduced. Influential design parameters are then selected based on their $\mu^*$ and $\sigma$ values or by visualization, as demonstrated in Section 4.5.

### 4.3.2 Design Objective Analysis

The randomized data set from the previous step is reused to estimate the distribution and correlation of design objectives. Without loss of generality, we consider minimizing all $m$ objectives. The vector consisting the minimum (best) of each output is called Utopia set which gives a lower bound of the Pareto optimal set, while the one consisting the maximum (worst) is Nadir set that gives the upper bound. As recommended in [45], these sets are determined directly from the data instead of optimizing each objective individually, whose computational cost is not affordable:

\[
\text{Utopia set } z_{i}^{U} := \min_{1 \leq j \leq k} f_i(x^{[j]}), \forall 1 \leq i \leq m
\]  \hspace{1cm} (4.3)

\[
\text{Nadir set } z_{i}^{N} := \max_{1 \leq j \leq k} f_i(x^{[j]}), \forall 1 \leq i \leq m.
\]  \hspace{1cm} (4.4)

We can also identify the correlations between design goals to reveal their mutual impact either visually or by calculating the correlation coefficients. As an example, the scattered plot
is used in Section 4.5. For score calculation, we recommend Spearman’s rank correlation coefficient[100] that assesses how strong a monotonic function describes the relationship between two variables. Suppose we have two goals $X$ and $Y$, after being converted to their respective ranks, denoted as $rg_X$ and $rg_Y$, the Spearman’s score is:

$$\rho_{(rg_X, rg_Y)} = \frac{cov(rg_X, rg_Y)}{\sigma_{rg_X} \sigma_{rg_Y}}.$$  \hspace{1cm} (4.5)

where, $cov(rg_X, rg_Y)$ is the covariance of the rank variables and $\sigma_{rg_X}, \sigma_{rg_Y}$ are their deviations.

### 4.4 Design Optimization

The most widely adopted global optimization technique for analog optimization is the stochastic search method[84], such as genetic algorithm and differential evolution, but the penalty is the high computational cost. Nevertheless, recently in various fields, such as robotics, automatic machine learning, and sensor networks, Bayesian optimization emerges as a powerful and sample-efficient global optimization solution[119].

In this section, we first illustrate the general idea of Bayesian optimization, then we introduce Bayesian linear regression that leads to Gaussian processes and Student-$t$ processes. In the end, we will describe several common acquisitions functions used in Bayesian optimization.
4.4.1 Bayesian Optimization Algorithm

As opposed to the evolutionary algorithms in which new samples are generated via crossover and mutation of the elite individuals in the population, Bayesian optimization (BO) utilizes the global information from a distribution model to sequentially generate new candidates. The BO flow contains two main parts: statistical surrogate model and acquisition function. The surrogate model represents the belief of the actual function which is updated iteratively with new samples, and the acquisition function uses the surrogate model to determine the next best sample to acquire. Therefore, they are intertwined to guide the sampling sequence to the optimal value by the Bayes' rule:

$$p(f|\mathbf{D}) = \frac{p(\mathbf{D}|f)p(f)}{p(\mathbf{D})}.$$  \hspace{1cm} (4.6)

Here, $f$ and $\mathbf{D}$ denote the surrogate model and the data respectively. Since $f$ is randomly distributed, we place a prior belief of the distribution $p(f)$, and it is updated at each iteration with $p(\mathbf{D}|f)$, the likelihood of data under the current function distribution. The acquisition function translates this likelihood to the next sample that is most probable to result in a better value, and the updated belief is then called the posterior $p(f|\mathbf{D})$. Notice the denominator $p(\mathbf{D})$ is the marginal likelihood of the data which is independent of $f$, and therefore, it becomes a normalizing constant.

There exists many efforts in the literature on multi-objective Bayesian optimization algorithms, such as in [18, 68]. However, consider the analog optimization situation where finding the whole Pareto set is not required, we utilize a simple yet effective scalarization technique presented in [45] which scales the objectives to a closed interval $[0, 1]$ with Nadir
and Utopia sets, and then take the sum to form a single objective. The objective function $f_{obj} : \mathbb{R}^k \rightarrow \mathbb{R}$ is:

$$f_{obj} = \sum_{i=1}^{m} \frac{f_i(x) - z_i^N}{z_i^U - z_i^N}.$$  \hspace{1cm} (4.7)

### 4.4.2 Bayesian Linear Regression

In Chapter 2, we reviewed the frequentist’s view of linear regressions. In particular, the assumptions about the model error (independence, homoscedasticity, and normality), the MLE estimate, regularization, and how to use cross validation to select the best model. In this section, we will switch to discuss the Bayesian approach.

The assumption of homoscedasticity assumes equal variance everywhere. Therefore our confidence (or equally uncertainty) about the model prediction is the same within the input space. This is not particularly realistic, as for unknown inputs in the vicinity of observed samples, the uncertainty is relatively small, while it becomes significantly higher in regions where no observed sample resides. Bayesian linear regression will enable us to incorporate this observation readily.

Extending the Bayes’ rule into the relationship between observed data $(D)$ and the model hypothesis $(h \in \mathcal{H})$, we have:

$$P(h|D) = \frac{P(D|h)P(h)}{P(D)} = \frac{P(D|h)P(h)}{\sum_{h'\in\mathcal{H}} P(d|h')p(h')}$$ \hspace{1cm} (4.8)
where \( P(h) \) is the prior belief of the model \( h \) before observing \( D \), \( P(D|h) \) is the likelihood of observing the data given the model, and \( P(h|D) \) is the posterior belief of \( h \) after \( D \) is observed. Therefore, Bayesian learning is essentially an iterative process of updating the belief of the model through observing more and more data.

From the last section, it is necessary to specify the prior, the likelihood, and the posterior for Bayesian linear regression. In particular, the prior and the posterior should be conjugate distributions with respect to the likelihood, which means the posterior distributions should be in the same family as the prior distribution[99]. Gaussian distributions are typical choices for Bayesian linear regression, due to the fact that they are self-conjugate given a Gaussian likelihood function, which we will show below.

One significant difference between frequentist’s and Bayesian interpretations lies in the treatment of model parameters: frequentists believe that there is a fixed set of model parameters that one needs to infer, while the Bayesian way treats them as random variables.

Our initial belief before seeing any evidence (data) of the model parameters is a Gaussian distribution with mean \( w_0 \) and variance \( \Sigma_0 \):

\[
P(w) = \mathcal{N}(w|w_0, \Sigma_0).
\] (4.9)

And the likelihood is \( \mathcal{N}(y|Xw, \sigma^2 I_n) \). We then calculate the posterior using the Bayes’
rule:

\[
P(w|X, y, \sigma^2) \propto \mathcal{N}(w|w_0, \Sigma_0) \mathcal{N}(y|Xw, \sigma^2 I_n)
\]

\[
\propto \exp\left[-\frac{1}{2\sigma^2}(y - Xw)^T(y - Xw)\right] \times \exp\left[-\frac{1}{2}(w - w_0)^T\Sigma^{-1}_0(w - w_0)\right]
\]

\[
= \exp\left[-\frac{1}{2}[(w - w_0)^T\Sigma^{-1}_0(w - w_0) + (y - Xw)^T\sigma^{-2}(y - Xw)]\right]
\]

\[
= \exp\left[-\frac{1}{2}(w^T\Sigma^{-1}_0 w - 2w_0^T\Sigma^{-1}_0 w + w_0^T\Sigma^{-1}_0 w_0 + \sigma^{-2} y^T y - 2\sigma^{-2} y^T Xw + \sigma^{-2} w^T X^T Xw)\right].
\]

(4.10)

By completing the squares, we further have:

\[
P(w|X, y, \sigma^2) \propto \exp\left[-\frac{1}{2}(w - w_n)^T\Sigma^{-1}_n(w - w_n)\right].
\]

(4.11)

where

\[
w_n = \Sigma_n (\Sigma^{-1}_0 w_0 + X^T y / \sigma^2)
\]

(4.12)

\[
\Sigma^{-1}_n = X^T (\sigma^2 I)^{-1} X + \Sigma^{-1}_n.
\]

(4.13)

The posterior probability should integrate to 1, from the right-hand side of Equation (4.11), using the property of multivariate Gaussian density function, we have:

\[
\int w \exp\left[-\frac{1}{2}(w - w_n)^T\Sigma^{-1}_n(w - w_n)\right]dw
\]

\[
= |2\pi \Sigma_n|^\frac{1}{2}.
\]

(4.14)
Therefore, replacing the "proportional to" with "equal to" in Equation (4.11) results in:

\[
P(w|X, y, \sigma^2) = \frac{1}{\sqrt{2\pi\Sigma_n}} e^{-\frac{1}{2}(w - w_n)^T \Sigma_n^{-1}(w - w_n)}
\]

(4.15)

\[
P(w|w_n, \Sigma_n).
\]

With Equation (4.15), we show the posterior is again a Gaussian distribution, and hence prove the self-conjugate property of Gaussian distributions.

### 4.4.3 Inference with Bayesian Linear Regression

Recall in Section 2.3.2 we were able to obtain the MLE estimate \( w_{MLE} \), the point estimate of the output \( y^* \) from an unseen input \( x^* \) is:

\[
y^*_MLE = x^*w_{MLE}.
\]

(4.16)

Adding the constant variance \( \sigma^2 \), the following distribution follows:

\[
P(y^*_MLE|x^*, X, y, \sigma^2; w_{MLE}) = \mathcal{N}(y^*_MLE|x^*w_{MLE}, \sigma^2).
\]

(4.17)

With Bayesian linear regression, the prediction distribution is instead given by marginalizing the posterior:

\[
P(y^*|x^*, X, y, \sigma^2; w) = \int w \mathcal{N}(y^*|x^*w, \sigma^2) \mathcal{N}(w|w_n, \Sigma_n) dw
\]

(4.18)

\[
= \mathcal{N}(y^*|x^*w_n, \sigma^2 + x^*T\Sigma_n x^*).
\]

where \( w_n \) and \( \Sigma_n \) are from Equations (4.12) and (4.13). Here we omit the procedures for
deriving the results above, and refer the interested readers to Chapter 4 in [99].

### 4.4.4 Comparison with Linear Regression

Comparing Equation (4.18) to (4.17), one notices the difference in the prediction variance: MLE gives an ubiquitous variance within the design space, while in Bayesian linear regression, the variance depends on the training set and the testing point.

On the other hand, assume the prior distribution of $\mathbf{w}$ has a zero mean and $\tau^2_0$ variance, $\mathcal{N}(\mathbf{w}|\mathbf{0}, \tau^2_0 \mathbf{I})$, using (4.12) and (4.13):

\[
\mathbf{w}_n = \Sigma_n (\mathbf{X}^T \mathbf{y} / \sigma^2)
= \frac{1}{\sigma^2} \left( \frac{1}{\tau^2} \mathbf{I} + \frac{1}{\sigma^2} \mathbf{X}^T \mathbf{X} \right)^{-1} \mathbf{X}^T \mathbf{y}
= (\lambda \mathbf{I} + \mathbf{X}^T \mathbf{X})^{-1} \mathbf{X}^T \mathbf{y}.
\]  

(4.19)

where $\lambda = \sigma^2 / \tau^2$. This indeed correspond to (2.25), therefore, Bayesian linear regression with a prior over the model parameter has exactly the same effect as the Ridge regression. To further reveal the rationale, if we have no prior knowledge about the model parameter, then $\tau^2 \to \infty$ and $\lambda \to 0$, and the posterior $\mathbf{w}_n$ will converge to that given by the MLE estimate.

### 4.4.5 Gaussian Processes

The core of Bayesian optimization is to have a surrogate model that gives high-quality mean prediction and prediction uncertainty. Bayesian linear regression fits this purpose for linear case, but to handle arbitrary cases, we need a more powerful nonparametric model. In this
and the next sections, we will discuss the Gaussian process and Student-\(t\) process models to be used in the BO settings.

Gaussian process processes, also known as Kriging models in the geostatistics and engineering literature, is an extension of multivariate Gaussian distribution\cite{33}. Instead of a distribution over random variables, Gaussian process defines a distribution over random functions. Precisely, a Gaussian process is an infinite collection of random variables such that any finite set of them follows a joint Gaussian distribution\cite{115}.

This distribution over random functions is completely characterized by its mean function \(m(\mathbf{x})\) and covariance function \(k(\mathbf{x}, \mathbf{x}')\):

\[
\begin{align*}
m(\mathbf{x}) &= \mathbb{E}[f(\mathbf{x})] \\
k(\mathbf{x}, \mathbf{x}') &= \mathbb{E}[(f(\mathbf{x}) - m(\mathbf{x}))(f(\mathbf{x}') - m(\mathbf{x}'))]
\end{align*}
\]

The notation that a function follows a Gaussian process is therefore,

\[
f(\mathbf{x}) \sim \mathcal{GP}(m(\mathbf{x}), k(\mathbf{x}, \mathbf{x}'))
\]

Any real-valued function can serve as the mean function \(m(\mathbf{x})\), however, for \(k(\mathbf{x}, \mathbf{x}')\) to be qualified as a Gaussian process covariance function, the matrix \(K\) in Equation (4.23) must be
a valid multivariate Gaussian distribution covariance matrix, or positive semi-definite [104].

\[
K = \begin{bmatrix}
k(x_1, x_1) & \ldots & k(x_1, x_m) \\
\vdots & \ddots & \vdots \\
k(x_m, x_1) & \ldots & k(x_m, x_m)
\end{bmatrix}
\] (4.23)

According to the Mercer’s condition [99], we call the function \( k(\cdot, \cdot) \) that makes \( K \) a covariance matrix a valid kernel function.

The most popular choice of a kernel function is the squared exponential (SE) kernel, also known as the Gaussian kernel or the Radial basis function (RBF) kernel, defined as:

\[
k_{SE}(x, x') = \exp\left(-\frac{1}{2l^2}||x - x'||^2\right)
\] (4.24)

Notice this has same formulation as the radial basis function (2.20), except that the center is no longer fixed.

Another popular choice of kernel function is the Matérn class, given by:

\[
k_{\text{Matérn}}(x, x') = \frac{2^{1-\nu}}{\Gamma(\nu)}\left(\frac{\sqrt{2\nu r_{xx'}}}{l}\right)^\nu K_\nu\left(\frac{\sqrt{2\nu r_{xx'}}}{l}\right)
\] (4.25)

where \( r_{xx'} = ||x - x'||, \nu > 0, l > 0 \) and \( K_\nu \) is a modified Bessel function [115]. Notice that as \( \nu \to \infty \), this class of kernels approach to the squared exponential kernel.

For the Matérn kernel class, Matérn 3/2 (\( \nu = 3/2 \)) and Matérn 5/2 (\( \nu = 5/2 \)) are the most interesting ones for machine learning, one of their advantages is being twice differentiable
without the smoothness requirement of the SE kernel. The Matérn 3/2 and 5/2 kernels are:

\[
k_{M3/2}(\mathbf{x}, \mathbf{x}') = \left( 1 + \frac{\sqrt{3}r_{\mathbf{x}\mathbf{x}'}}{l} \right) \exp \left( -\frac{\sqrt{3}r_{\mathbf{x}\mathbf{x}'}}{l} \right) \tag{4.26}
\]

\[
k_{M5/2}(\mathbf{x}, \mathbf{x}') = \left( 1 + \frac{\sqrt{5}r_{\mathbf{x}\mathbf{x}'}}{l} + \frac{5r_{\mathbf{x}\mathbf{x}'}}{3l^2} \right) \exp \left( -\frac{\sqrt{5}r_{\mathbf{x}\mathbf{x}'}}{l} \right) \tag{4.27}
\]

Both the SE and Matérn kernels have learnable length scale hyper-parameters \(l\), this class of kernel are commonly referred to as \textit{automatic relevance determination (ARD)} kernels. One nice property of ARD kernel is that the length scale parameter serves as a metric for dimensionality reduction\[115\], a larger \(l\) indicates a less important feature.

Aside from the SE and Matérn kernels, there exists other kernels which are powerful in certain applications. For instance, the periodic kernel (for periodic or time-series problems) and the additive kernels (for high dimensional regressions). In \[28\], the author presented an exhaustive review of how to construct and apply kernel functions in Gaussian processes.

\section*{4.4.5.1 Gaussian Processes for Regression}

Suppose the current training data set is \(D = \{\mathbf{x}_i, f_i\}_{i=1:n}\), given a new input \(\mathbf{x}^*\), the joint distribution is hereby updated:

\[
\begin{bmatrix} f \\ f^* \end{bmatrix} \sim \mathcal{N} \left( \begin{bmatrix} \mu([\mathbf{x}^*]) \\ \mathbf{k}^T \end{bmatrix}, \begin{bmatrix} \mathbf{K} & \mathbf{k} \\ \mathbf{k}^T & k(\mathbf{x}^*, \mathbf{x}^*) \end{bmatrix} \right) \tag{4.28}
\]

where \(\mathbf{k}^T = [k(\mathbf{x}^*, \mathbf{x}^1), ..., k(\mathbf{x}^*, \mathbf{x}^n)]\).

For simplicity, we assume \(\mu_0(\mathbf{x}_i) = 0\), so the posterior distribution after incorporating \(\mathbf{x}^*\)
in a *noiseless* setting is:

\[ f^* \sim \mathcal{N}(\mu(x^*), \sigma^2(x^*)) \quad (4.29) \]

the predicted mean and covariance functions are:

\[ \mu(x^*) = k^T K^{-1} f \quad (4.30) \]
\[ \sigma^2(x^*) = k(x^*, x^*) - k^T K^{-1} k \quad (4.31) \]

However, in a *noisy* situation, the Gaussian processes are not required to "interpolate" the data, instead, they are required to be close to the observed data[99]. By assuming additive *i.i.d.* Gaussian noise with variance \( \sigma^2_n \), the \( K \) is modified by adding \( \sigma^2_n \) along the diagonal: \( K + \sigma^2_n I \). The predictions are thus:

\[ \mu(x^*) = k^T (K + \sigma^2_n I)^{-1} f \quad (4.32) \]
\[ \sigma^2(x^*) = k(x^*, x^*) - k^T (K + \sigma^2_n I)^{-1} k \quad (4.33) \]

### 4.4.6 Student-*t* Processes

Gaussian process model is the most popular surrogate model for Bayesian optimization. However, the GP model is less robust when inferring with heavily tailed noise[128]. Shah[118] showed that Student-*t* process (TP) model as an alternative to GP is better suited for this scenario.

Both Gaussian and Student-*t* are elliptical processes, but Student-*t* is a more general
one. Suppose \( f \sim \mathcal{TP} (\nu, \phi, \mathbf{K}) \), as the degree of freedom \( \nu \in \mathbb{R}_+ \setminus [0, 2] \to \infty, \mathcal{TP} (\nu, \phi, \mathbf{K}) \to \mathcal{N}(\phi, \mathbf{K}) \), where \( \phi \) is the mean function and \( \mathbf{K} \) the kernel.

TP models is characterized by the density function[118]:

\[
p(f) = \frac{\Gamma\left(\frac{\nu+n}{2}\right)}{\Gamma\left(\frac{\nu}{2}\right)\left((\nu-2)\pi\right)^{n/2}}|\mathbf{K}|^{-1/2} \\
\times \left(1 + \frac{(y-\phi)^T \mathbf{K}^{-1}(y-\phi)}{\nu-2}\right)^{-\frac{n+n}{2}}
\]

Similarly, suppose \( \phi = 0 \) for simplicity, the predictions made by TP of an unseen data \( \mathbf{x}^* \) from \( n \) observed noise-free samples are:

\[
f^* \sim \mathcal{TP} (\nu + n, \phi^*, \frac{\nu + \beta - 2}{\nu + n - 2} \mathbf{K}^*)
\]

where \( \phi^* = \mathbf{k}^T \mathbf{K}^{-1} \mathbf{f} \)

\[
\beta = \mathbf{f}^T \mathbf{K}^{-1} \mathbf{f}
\]

\[
\mathbf{K}^* = \mathbf{k}(\mathbf{x}^*, \mathbf{x}^*) - \mathbf{k}^T \mathbf{K}^{-1} \mathbf{k}
\]

where the notations follow those in Equation 10. Therefore, the predicted mean is \( \phi^* \) and variance is \( \sigma_{\tilde{n}}^2 = \frac{\nu + \beta - 2}{\nu + n - 2} \mathbf{K}^* \).

The prediction made from noisy samples are similarly obtained by including \( \sigma_{\tilde{n}}^2 \mathbf{I} \) along the diagonal of \( \mathbf{K} \), we skip the derivation to avoid redundancy. By comparing Eq. (4.29) and Eq. (4.35), one notice the key difference in the predictive covariances: the covariance of TP depends on the training data while that of GP does not. This improvement in predictive covariance is a critical factor for the Bayesian optimization performance.
4.4.7 Acquisition Functions

Both GP and TP regressions predict the mean value (Eq. 4.18, 4.20 & 4.24) and its uncertainty (Eq. 4.19, 4.21 & 4.26) of an unobserved data. Consider a minimization problem, where the lowest value could exist in regions with lower means or higher variances. The former exploits the current best data while the latter explores the unseen space. The mechanism for picking the next sample is thus trading off exploitation and exploration. Precisely, we express this trade-off with a utility function $U: \mathbb{R}^d \times \mathbb{R} \times \Theta \to \mathbb{R}$ that maps the input data, output data, and their associated function hyper-parameters into a scalar value, where a higher function value indicated a more "promising" point. With this mapping, the problem of trading off exploration and exploitation is transformed into another optimization problem of finding the maximum acquisition function output. This optimization problem is simpler in that (1) we have an explicit mathematical expression and (2) the calculation is entirely based on the surrogate model which is cheap to evaluate, so that many traditional optimization algorithms, such as DIRECT[90] or BFGS algorithms[5], becomes efficiently accessible.

Many criteria are introduced over the years, such as probability of improvement (PI), expected improvement (EI), lower/upper confidence bound (L/UCB), and the portfolio of multiple criteria[119]. In this section, we will take a brief survey of them.

4.4.7.1 Probability of Improvement

This criteria characterize the probability that the next evaluation point will be better than the observed best point. Suppose we use Gaussian processes as the surrogate model, from the previous section, we know that the posterior conditioned on any given point is a univariate Gaussian. With the current best observation being $\tau$, we can analytically calculate the
probability that the result of an unobserved point being better than $\tau$ as:

$$U_{PI}(x^*; X, Y) = P(y^* > \tau) = \Phi\left(\frac{\mu(x) - \tau}{\sigma_n(x)}\right)$$

(4.39)

where $\Phi$ is the standard normal cumulative distribution function.

When using Student-$t$ processes, the PI criteria is written as:

$$U_{PI}(x^*; X, Y) = \Lambda_{\nu+N}\left(\frac{\mu(x) - \tau}{\sigma_n(x)}\right)$$

(4.40)

where $\Lambda_{\nu+N}$ is the cumulative distribution function of standard T distribution with $\nu$ degree of freedom.

4.4.7.2 Expected Improvement

Intuitive as the probability of improvement (PI) seems, it actually performs very well when there exists a target optimum (such as the specification of designs). However, when no such information is available, the PI criteria tends to overly exploit[119]. In this case, instead of characterizing the probability, one usually turns to the expectation. Such criteria that analytically calculate how much improvement is expected against the best observed value is called expected improvement (EI).

Jones[62] first popularized the EI criterion. Again, suppose the current best result is $\tau$, the improvement function is:

$$\Delta(x) := (f(x) - \tau)^I_{f(x) > \tau}$$

(4.41)
The EI is then:

\[ E(\Delta) = Z \sigma(x) \Phi(Z) + \sigma(x) \phi(Z) \] (4.42)

where, \( Z = (\mu(x) - \tau)/\sigma(x) \), and \( \Phi(Z) \) and \( \phi(Z) \) are the CDF and PDF of standard normal distribution, respectively.

While in the Student-\( t \) process setting, we can show the EI as:

\[ E(\Delta) = Z \sigma(x) \Lambda_{\nu,N}(Z) + \sigma(x)(1 + \frac{Z^2 - 1}{\nu + N - 1}) \lambda_{\nu,N}(Z) \] (4.43)

where \( \Lambda(Z) \) and \( \lambda(Z) \) are the CDF and PDF of standard T distribution with \( \nu \) degree of freedom.

### 4.4.7.3 Lower/Upper Confidence Bounds

The original upper (and equivalently lower) confidence bound (UCB / LCB) idea dates back to [77], and the principle behind this criteria is to treat uncertainty optimistically. The UCB at point \( x^* \) is expressed as:

\[ U_{UCB}(x^*) = \mu(x^*) + \beta \sigma(x^*) \] (4.44)

where both \( \mu(x^*) \) and \( \sigma(x^*) \) are predicted with Gaussian or Student-\( t \) processes.

The \( \beta \) parameter is the trade-off weight of mean and variance, so that a larger \( \beta \) favors exploration (uncertainty) more than exploitation. A recent work in [124] discovers that the annealed UCB, in which \( \beta \) decays with iterations, can usually lead to a faster convergence and performance boost.
4.5 Design Examples

In this section, we will demonstrate the efficacy of our proposed approach using two 77GHz circuit blocks, including a balun and a power amplifier, which were manually designed for a previous tape-out. They are first analyzed and redesigned in the original process node (IBM 8XP), and then reused in a more advanced node (IBM 9HP) in this demonstration.

4.5.1 Test Case 1: 77GHz Balun

An LC balun is a common passive component in RFIC that converts a signal into a pair of out-of-phase ones or vice versa while suppressing the common-mode signal on the output. Simple as it seems, the balun design is an effective test case due to the interactions of variables and mutual impact of design objectives, i.e. the multi-port S-parameters. We show the schematics in Figure 4.2(a).

We first analyze the design for parameter screening and design objective visualization. The results are shown in Figure 4.2(b) and 4.3, respectively. As one can see in Figure 4.2(b), the lengths of $I_4$ and $I_1$ along with the length and width of $C_1$ show the most influence while the spacing in $I_1$ and its width contributes a lot less; Fig. 3 visualize the design objective analysis results, for example, Goal 5 and Goal 3 are positively related thus one improves with the other. On the other hand, Goal 4 correlates with Goal 2 and Goal 1 in a piecewise fashion, mostly due to the S-parameter phase wrapping trait.

The design optimization step is then executed. We compared the performances of the Genetic algorithm (GA) as an example of evolutionary algorithms and Bayesian optimization using GP model (BO-GP) and TP model (BO-TP). Figure 4.4 shows the function
evaluation iterations. One can obviously see the advantage of BO over GA: to reach the same objective function value (the red dashed line), GP-BO is over three times, and TP-BO is over eight times more sample-efficient than GA.

We plot the final design result in Figure 4.5. Note that the IP redesign in 8XP is improved from the manual one in many ways: $S_{11}$ and $S_{22}$ are placed closer to each other, while $S_{23}$ and $S_{13}$ are more symmetric about the center. The loss introduced by $S_{23}$ and $S_{13}$ are also lower. This design is then reused in IBM 9HP node with the same configurations, including design parameters and optimization setup, due to the technology similarity. The Smith chart for the reused balun is in Figure 4.5(c), and we observe a further improvement in the $S_{11}$-$S_{22}$ alignment.

Figure 4.2 (a) Balun schematics and (b) parameter screening
Figure 4.3 Design goal distribution and correlation
Figure 4.4 Balun optimization comparison
Figure 4.5 (a) Manual balun design in 8XP; (b) Balun redesign in 8XP; (c) Balun reuse in 9HP
4.5.2 Test Case 2: 77GHz PA Design

A power amplifier (PA) generates an amplified input signal to feed the antenna. In this test case, we concentrate on the design of the matching networks that ensure the optimal impedance in the transistors' working frequency range. Figure 4.6 shows the schematics of the (single-ended) PA design.

We first analyze the design and screen the 17 design parameters, as shown in Figure 4.7. The problem dimension is largely reduced as only 7 of them prove to be significant. Design objective analysis is omitted since the only goal in this PA design is to maximize the sum of power amplifier efficiency (PAE) and the corresponding output power (Pout). In the optimization stage, we also compare the performances of GA, BO-GP and BO-TP for the PA problem (Figure 4.8): for the same objective function value, GA consumes four times more samples than BO-GP and BO-TP. Similar to the balun case, BO-TP also outperforms the BO-GP algorithm.

We show the results for manual design and IP redesign in 8XP along with the IP reuse in
Figure 4.7 PA Design parameter screening
Figure 4.8 PA optimization comparison
Figure 4.9 (a) Manual PA Design in 8XP; (b) PA redesign in 8XP; (c) PA reuse in 9HP; (d) Result comparisons (Simulated at 80GHz)

<table>
<thead>
<tr>
<th></th>
<th>peakPAE (%)</th>
<th>Psat (dBm)</th>
</tr>
</thead>
<tbody>
<tr>
<td>8XP man.</td>
<td>20.9</td>
<td>14.8</td>
</tr>
<tr>
<td>8XP BO</td>
<td>21.8</td>
<td>14.6</td>
</tr>
<tr>
<td>9HP BO</td>
<td>23.5</td>
<td>18.3</td>
</tr>
</tbody>
</table>
9HP in Figure 4.9. From 4.9(a) and 4.9(b), the PAE and Pout in the redesign are higher than the original manual one, and yet another boost is achieved after reusing the IP in IBM 9HP (4.9(c)). This is consistent with the improvement claimed in the 9HP technology.

4.6 Conclusions

We present the methodology to automatically analyze, redesign and reuse analog IPs based on statistical learning techniques. The flow efficacy is verified with two 77GHz circuit blocks across two different technology nodes. Results indicate that the proposed flow can automatically analyze and produce satisfactory designs with minimal human effort, while consuming 3 to 8 times less computational power than traditional genetic algorithm-based approaches. Moreover, to the best of the authors’ knowledge, this is the first engineering design application of Bayesian optimization using the Student-$t$ process model. Our demonstrations indicate the advantages of Student-$t$ process model over Gaussian process model in both efficiency and solution quality of Bayesian optimization.
In Chapter 2, we surveyed on the rare-even Monte Carlo simulation techniques and discussed the advantages of using predictive models to assist the sampling process when handling expensive simulations. In particular, the work presented by Trent et al.[93] introduced the high-sigma Monte Carlo (HSMC) algorithm which uses symbolic regression models to reorder sample sequence. HSMC significantly speed up the convergence of the
sampling-based Monte Carlo by placing samples that are more prone to fail to be simulated first.

In this chapter, we will show how to use ensemble models to improve the HSMC framework in the overall performances including predictions accuracy and convergence speed without losing the merit of scalability. In the next sections, we will first review the HSMC algorithm and the Fast Function Extraction (FFX) surrogate model used in the framework. Then we discuss how ensemble models can enhance the algorithm’s performance in both prediction speed and accuracy. Finally, we verify our the claims using both academic structural reliability examples and an industrial 6T SRAM cell design.

5.1 High Sigma Monte Carlo (HSMC) algorithm

The fundamental idea of the HSMC algorithm is to mine the large Monte Carlo candidate samples using predictive models. As opposed to rejection sampling\[116\] or the statistical blockade algorithm\[122\], HSMC uses the predicted results to rank the simulation sequence but doesn’t completely rule out candidate samples, this property makes it more resilient to errors due to surrogate model inaccuracy.

The HSMC algorithm outline is summarized in Algorithm 1:

5.1.1 Symbolic Regression and FFX

The HSMC algorithm’s efficacy relies on the symbolic regression surrogate model which scales well with higher dimensionality. We will briefly discuss the Geometric Programming

\[\text{Presented in this chapter is the work accomplished during his internship at Samsung Device Lab}\]
Algorithm 1: HSMC Algorithm

1 **Initialization:**
2 Generate \( N_{MC} \) candidate Monte Carlo samples based on the required sigma level;
3 Construct initial DoE with \( N_{init} \) samples farthest from nominal;
4 Simulate initial samples and build the training set: \( \{X_{train}, y_{train}\} = \{X_i, y_i\}_{i=1,...,N_{init}} \);
5 Count failure samples in the initial training set: \( N_f = \sum_{i=1}^{N_{init}} 1_{y_i > u} \);
6 **while** convergence condition not met **do**
7 Train the surrogate model with the training set;
8 Use the surrogate model to predict unobserved Monte Carlo candidates;
9 Order the Monte Carlo candidates from worst case predictions to the best ones;
10 Starting from the worst ones, simulate \( N_{add} \) samples;
11 Update the failure count: \( N_f = N_f + \sum_{i=1}^{N_{add}} 1_{y_i > u} \);
12 Update the training set with newly simulated samples;
13 **end**
14 The final (point) estimated failure rate \( p_f = \frac{N_f}{N_{MC}} \);
15 The final estimated confidence interval is calculated with Wilson’s score[103];

(GP) based symbolic regression model and the improved Fast Function Extraction (FFX) algorithm. For an in depth discussion, we refer the reader to [92].

5.1.1.1 Symbolic Regression via Genetic Programming

Symbolic regression (SR) is a way of fitting measured/experimental data into a set of mathematical equations, for instance, “\( x^n + a \)”, “\( \cos(b \cdot x) \)”, or “\( c / e^x \)”, etc. Equation fitting was originally done manually until Koza[70] first introduced the idea of using genetic programming (GP) to automate this process, we call this method genetic programming symbolic regress (GP-SR) for short. The main idea of GP-SR is to iteratively select the functional set and the terminal set in each evolution step with crossover and mutations of the tree structure. The functional set refers to the operators while the terminal set consists
either variables or constants. As an illustration, consider the equation:

\[ y = x_1 + (x_2 - 3.1) \times x_1 \]

It can be expressed in a tree structure (Figure 5.1):

The evolutions process get executed as such:

1. Randomly pick the operators, variables, and constants in a top-to-bottom fashion, the process ends when there are no more operators on the terminal nodes. With the same procedure, more trees are generated, the generation size is typically on the order of 500 trees [70]

2. Evaluate the fitness of each tree by measuring the prediction error and select the top individuals as the parents for the next generation

3. Generate new trees through crossovers and mutations or the tree branches
4. Repeat Step (2) – (4) until desired accuracy is reached

GP-SR are capable of scaling to high dimensionality and handle nonlinearity (using nonlinear operators, for instance, the step function). However, the evolutionary training process is too time-consuming that restricts its broader applications.

5.1.1.2 Fast Function Extraction (FFX)

To overcome the large training overhead of GP-SR, [92] proposed to use pathwise regularized learning techniques to prune the basis and identify the best coefficients for the symbolic regressors.

Essentially, pathwise regularized learning minimizes the elastic net formulation of regularized cost function (Equation 5.1) using the coordinate descent[36] algorithm:

$$a^* = \arg\min_a (||y - Xa||_2 + \lambda_2 ||a||_2 + \lambda_1 ||a||_1)$$  \hspace{1cm} (5.1)

The main steps of FFX are listed below:

1. Generate a large set of basis functions that combine one or two nonlinear sub-functions

2. Use pathwise regularized learning to identify the best coefficients and bases when there are 0, 1, 2 bases and so on

3. Filter the candidate functions to a non-dominated set (Pareto set) that trades off number of bases and error
FFX inherits the flexibility of GP-SR while significantly speed up the training time, as [92] argues, FFX has the speed of linear models while the accuracy is comparable to GP-SR, this is the main advantage that makes FFX suitable for the HSMC flow.

However, despite the appealing characteristics of FFX model, there are two main issues: (1) The accuracy of FFX depends on the quality of the “model factory” (i.e. candidate basis functions), this limits its strength to become a universal approximator, and (2) FFX is a deterministic model that can’t produce the confidence interval associated with a point estimate. In our application, however, we not only require the models to assist in yield estimation, but also need to use them in the follow-on yield aware optimization task, therefore the prediction confidence is important. For these reasons, we propose to improve the HSMC framework with ensemble learning models, i.e. gradient boosting and random forest.

5.2 Ensemble Models

Ensemble learning forms a stronger “committee” with many weak models[36]. This formation implies a desirable advantage: the combination of weak learners will average out the estimation bias while suppressing the chance of overfitting without the loss of flexibility. Therefore, ensemble models usually achieve the best of both worlds in trading off bias and variance. In this section, we will introduce two powerful ensemble learning algorithms, called gradient boosting and random forest, and incorporate them into the predictive model enabled Monte Carlo simulation flow.
5.2.1 Gradient Boosting

Boosting is a type of constructive ensemble learning, which formulates stronger predictors through sequentially adding new models to compensate for the residue errors from previous predictors. Original boosting models include AdaBoost[34], LogitBoost[35], and so on. Beiman et al.[37] first introduced the idea of gradient boosting. He observed that the process of sequentially fitting new models resembles that of the gradient descent algorithm, but in the functional space.

Gradient descent, also known as steepest descent, is a first-order iterative optimization algorithm that finds the (local) optimum of a function by iteratively taking steps along the fastest descending gradient (the negative gradient) from the current evaluation point.

Suppose the loss function is $J(\theta)$, gradient descent updates the next evaluation point $\theta_{i+1}$ as such:

$$
\theta_{i+1} = \theta_i - \alpha \frac{\partial J}{\partial \theta_i}.
$$

where $\alpha$ is called the learning rate.

Adopting the similar idea but moving to the functional space, we can redefine the loss function:

$$
L(F(x)) = \frac{1}{2}(y - F(x))^2.
$$

Here, $y$ is the true observation and $F(x)$ is the predictor. Also notice that we use squared loss for the sake of brevity of discussion, and this easily extends to other choices, such as the absolute loss or the Huber loss. Interested readers are directed to [101] for an in-depth
discussion of other loss function choices.

The objective function that we would like to optimize is defined as the sum of prediction errors from $F(x)$ [81], mathematically:

$$J(F(x)) = \sum_i L(F(x_i)). \quad (5.4)$$

Then a similar updating strategy for $F(x)$ is obtained:

$$F(x_{i+1}) = F(x_i) - \frac{\partial J(F(x_i))}{\partial F(x_i)}. \quad (5.5)$$

With Equation (5.3) & (5.4), we further have:

$$\frac{\partial (J(F(x_i)))}{\partial (F(x_i))} = \frac{\partial \sum_i L(F(x_i))}{\partial F(x_i)} = F(x_i) - y_i. \quad (5.6)$$

This way, we show that updating $F$ based on residue error (called *boosting*) is directly correlated to updating the evaluation point based on the negative gradient in the gradient descent algorithm.

It should also be mentioned that along with the choices of loss functions, gradient boosting also offers flexible choices of the base learner models. In this work, we will resort to the regression tree as our based learner, but other models are also demonstrated in [101]. Gradient boosting adjust the quantile loss function (see [16]) to characterize the prediction confidence interval.

The complete gradient boosting algorithm is listed in Algorithm 2.
Algorithm 2: Gradient Boosting Algorithm

1. **Inputs:**
   1. Input data \( \{X_i, y_i\}_{i=1}^N \)
   2. Number of iterations \( M \)
   3. Loss function \( L(y, f) \)
   4. Base learner model \( h(x, \theta) \)

2. **Main Algorithm:**
   1. Initialize \( \hat{f}_0 \) with a constant
   2. for \( i \leftarrow 1 \) to \( M \) do
      3. Compute the negative gradient \( g_i(x) \)
      4. Fit a new base learner \( j(x, \theta_i) \)
      5. Determine the best step size \( \alpha_i \):
         \[ \alpha_i = \arg\min_\alpha \sum_{t=1}^N L(y_t, \hat{f}_{i-1}(x_t) + \alpha h(x_t, \theta_i)) \]
      6. Update the function estimate:
         \[ \hat{f}_i \leftarrow \hat{f}_{i-1} + \alpha h(x, \theta_i) \]
   3. end

5.2.2 Random Forest

5.2.2.1 Decision Tree Regression

We begin our discussion on the random forest model by introducing the decision tree regressor. Decision trees are typically used as classifiers, where each internal node recursively split the input domain into different branches using the feature thresholds. The splitting is done by minimizing certain metrics, such as the Gini index or the entropy gain. After training, the leaf nodes of a decision tree classifiers represents the class labels, predicting an unseen data, therefore, involves traversing the decision tree from the root to one of the leaf nodes. When used as a regressor, the decision tree regressor differs from the classifiers in two aspects: the outputs are real-valued instead of discrete, and the splitting criteria are
Breiman introduces one of the most widely used training method for a decision tree regressor, called Classification and Regression Trees (CART)\cite{12}. In brief, the CART algorithm takes a greedy binary search approach to find the best split among all the distinct features by minimizing the sum of squared error (SSE). Suppose we consider splitting variable \( i \) at the threshold of \( s \), this node partitions the dataset into two subsets: \( S_1 = \{ X | X_i < s \} \) and \( S_2 = \{ X | X_i \geq s \} \). The goal is to seek the splitting variable \( i \) and threshold \( s \) such that the following criteria (SSE) are at its minimum\cite{36}: 

\[
SSE(i, s) = \sum_{x_i \in S_1} (y_i - c_1)^2 + \sum_{x_i \in S_2} (y_i - c_2)^2.
\] (5.7)

where \( c_1 \) and \( c_2 \) are the mean value of the outputs in each subsection:

\[
c_1 = \text{avg}\{ y_i, | x_i \in S_1 \}
\]

\[
c_2 = \text{avg}\{ y_i, | x_i \in S_2 \}.
\] (5.8)

This greedy approach will continue to split until there is only one sample at each node, this results in overfitting. As a prevention, the tree is pruned by adding the penalization term in the cost function:

\[
SSE_p(i, s) = \sum_{x_i \in S_1} (y_i - c_1)^2 + \sum_{x_i \in S_2} (y_i - c_2)^2 + \alpha |T|.
\] (5.9)

where \( |T| \) is number of nodes in the decision tree \( T \).
5.2.2.2 Random Forest

Compared to the gradient boosting models, random forest takes a polling approach to formulate the ensemble models. As the name suggests, a random forest model consists of many decision trees, and each tree is trained on the bootstrapped sample (drawn with replacement) from the original training set. Another randomness is introduced in the node-splitting step, unlike decision trees in which nodes are split based on the best criterion (entropy gain or Gini index for classification trees and SSE for regression trees) among all features, random forest splits the nodes on a random subset of the features. Though this will cause an increase of bias in each individual learners, random forest overcome it by averaging when making the final decision.

The overall steps of building a random forest are illustrated in Algorithm 3:

**Algorithm 3**: Random Forest for Regression

1. **for** $i \leftarrow 1$ **to** $B$ **do**
2. Draw a bootstrap sample $Z$ of size $N$ from the training data
3. Train a random forest tree $T_b$ with $Z$ with the following steps until each node contains no more than $n_{min}$ samples:
   1. Select $m$ features at random from $n$ total features (Using Equation (5.9))
   2. Pick the best variable to split and split the node into two sub-nodes
4. **Output** the ensemble of tree $T_{b=1,\ldots,B}$

For a new data point $x^*$, the predicted value is:

$$y^* = \frac{1}{B} \sum_{b=1}^{B} T_b(x).$$  \hspace{1cm} (5.10)
Suppose we have a forest of 1000 trees, the equation above is essentially averaging 1000 predictions from different decision trees. Statistically, this is called the sample mean or point estimation. Further more, we can also use these 1000 predictions to calculate the confidence interval associated with this point estimate empirically. This information is especially useful for quantifying prediction uncertainty.

Besides the predictive power for mean and variance, the random forest can also select important features by evaluating the relative rank (depth of the node) of the variables. This property is critical when dealing with higher dimensional design problems. Generally speaking, more important features reside in the shallower levels in a random forest model, as the higher level nodes contribute to the final prediction decision of a larger fraction of the input samples[109].

5.3 Implementation Details

In this section, we discuss the implementation details of the Python package called Surrogate Model assisted Monte Carlo (SMMC).

5.3.1 Implementation Concerns

The SMMC package aims at efficiency and general applicability. One major concern is there for the turn-around time (TAT).

As Equation (1.6) shows, to achieve 10% relative error, the sample number needs to be $100/\alpha f$. Suppose we are to verify a 6−σ design, empirically we need to generate > 5 billion Monte Carlo samples in the first step, which imposes a large overhead cost for the predicting
Table 5.1 Model parameter settings

<table>
<thead>
<tr>
<th>Model Type</th>
<th>Major Parameter Settings</th>
</tr>
</thead>
<tbody>
<tr>
<td>GP</td>
<td>kernel: RBF, $\alpha$: 1e-10, optimizer: L-BFGS</td>
</tr>
<tr>
<td>LR</td>
<td>fit intercept: true, $\alpha$: picked with 10-fold CV</td>
</tr>
<tr>
<td>NN</td>
<td>hidden layer size: 100, activation: ReLU, solver: Adam, $\alpha$: 0.0001, max iteration: 200</td>
</tr>
<tr>
<td>GB</td>
<td>loss function: least square, n_estimators: 100, learning rate: 0.1, max_depth: 3</td>
</tr>
<tr>
<td>RF</td>
<td>n_estimators: 10, criterion: MSE</td>
</tr>
<tr>
<td>FFX</td>
<td>N/A</td>
</tr>
</tbody>
</table>

and sorting steps. To solve these potential issues, we propose the following solutions.

5.3.2 Predictive Model Selection

The prediction speed is largely determined by the surrogate model type, simple models like linear regression will be much faster than nonparametric models, such as Gaussian processes. In this section, we will conduct a comparative study of the prediction speed of various candidate models, including the FFX model used in the original HSMC algorithm.

The surrogate models under consideration include: (1) Gaussian processes (GP), (2) (Ridge) linear regression (LR), (3) neural networks (NN), (4) gradient boosting (GB), (5) random forest (RF), and (6) FFX. The models’ major hyper-parameter setting are listed in Table 5.1, and we adopt the scikit-learn implementations[109].

To test the prediction speed, we prepare 10 million randomly generated 2-D input data to be fed into the surrogate models and repeat the process 10 times. The results are shown in Table 5.2.

From these results, one immediately notices the huge computational cost of Gaussian process model prediction. Therefore Gaussian process model is not practically feasible to
Table 5.2 Prediction Speed Comparison (10^7 predictions)

<table>
<thead>
<tr>
<th>Model Type</th>
<th>mean (s)</th>
<th>std (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>GP</td>
<td>60</td>
<td>0.2</td>
</tr>
<tr>
<td>LR</td>
<td>0.07</td>
<td>0.0024</td>
</tr>
<tr>
<td>NN</td>
<td>6.64</td>
<td>0.045</td>
</tr>
<tr>
<td>GB</td>
<td>11.1</td>
<td>0.0069</td>
</tr>
<tr>
<td>RF</td>
<td>7.57</td>
<td>0.021</td>
</tr>
<tr>
<td>FFX</td>
<td>2.61</td>
<td>0.0052</td>
</tr>
</tbody>
</table>

Table 5.3 Cantilever Parameter Distribution

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Distribution</th>
<th>mean</th>
<th>std</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x_1$</td>
<td>normal</td>
<td>$10^{-3}$</td>
<td>$2 \times 10^{-4}$</td>
</tr>
<tr>
<td>$x_2$</td>
<td>normal</td>
<td>250</td>
<td>37.5</td>
</tr>
</tbody>
</table>

be used in the proposed flow. Moreover, for dealing with more difficult cases, it is ideal for the surrogate model prediction speed to remain constant with an increasing data size. This is another reason why we can't use GP in the framework, despite its appealing characteristic of confidence interval prediction.

However, being fast needs not sacrifice the predictive capability. In this test, we examine the accuracy of the models using the 2-D cantilever problem[9]:

$$f(x_1, x_2) = 18.46154 - 7.476923 \times 10^1 \frac{x_1}{x_2^3}. \quad (5.11)$$

with the two variables $x_1$ and $x_2$ normally distributed (Table 5.3).

The cantilever problem is a well studied structural reliability test case, known for its difficulty to estimate its yield efficiently. We will also use it as one of the benchmarks to test the SMMC package in the upcoming "Testing Result" section, but for now, this is simply a deterministic equation that we want to model.
Table 5.4 Prediction accuracy comparison

<table>
<thead>
<tr>
<th>Model Type</th>
<th>mean (MSE)</th>
<th>std (MSE)</th>
</tr>
</thead>
<tbody>
<tr>
<td>LR</td>
<td>3.23</td>
<td>0.618</td>
</tr>
<tr>
<td>NN</td>
<td>24.10</td>
<td>0.045</td>
</tr>
<tr>
<td>GB</td>
<td>0.298</td>
<td>0.385</td>
</tr>
<tr>
<td>RF</td>
<td>0.236</td>
<td>0.140</td>
</tr>
<tr>
<td>FFX</td>
<td>0.874</td>
<td>0.803</td>
</tr>
</tbody>
</table>

The accuracy metric in this test is the MSE error. We randomly pick 1000 sample amongst the 10 million samples in the previous step and test them with "70% train - 30% test" cross validation approach. Note that as FFX returns a non-dominant Pareto set of models, the "median" in this model set is selected for evaluation, as it represents the best trade-off between model complexity and prediction accuracy.

In Table 5.4, we demonstrate the result of each model, note we don't proceed with GP anymore due to its apparent speed drawback.

These results demonstrate the predictive ability of ensemble models (RF and GB). Especially, consider the trade-off between prediction speed and accuracy, random forest achieves 1/4 prediction errors at the cost of extra 5 second prediction time for 10 million predictions compared with the FFX, without losing the capability of modeling high-dimensional and nonlinear problems. Along with the nondeterministic characteristics, ensemble models, especially random forest, are qualified to be a candidate model besides FFX in the framework.
5.3.3 TAT Optimization

Besides the prediction speed and accuracy bottleneck, sorting the massive predicted Monte Carlo samples impose another challenge for the total turn-around time (TAT).

Recall from Algorithm 1, we need to order the MC candidates from the worst to the best-predicted result and pick the top $N_{add}$ sample to simulated in the next iteration. Typical sorting algorithms (such as merge sort or quick sort) has $O(n \log(n))$ time complexity, this is infeasible for billion-scale tasks. To solve this issue, we used multi-threading and fixed-size max-heap.

In the multi-threading step, we employ a divide-and-conquer strategy and partition to MC samples into $N_{thread}$ chunks to perform the prediction and sorting steps on each thread, the result is then combined in the main thread.

However, in each sorting step, as a strict order of the samples is not required, we use a fixed-size ($N_{add}$) max-heap to accelerate the sample picking process. For a $N_{add}$-size heap, the heapify step has $O(\log(N_{add}))$ time complexity, therefore the total expected time complexity is still $N_{add}O(\log(N_{add}))$. During the test, we found that the average time for processing $10^7$ data is 30.3 seconds. Consider the result in Table 5.4, this is almost 6-10 times of the prediction time. To further optimize this issue, we resort to the order statistics of the predicted samples and calculate the 95 percentile cut-off threshold. After this one-time up-front step, we block 95% of the the heapify operations and eventually reduce the ordering cost to an average of 2.4 seconds.
5.4 Testing results

In this section, we test the efficacy of the proposed predictive model assisted Monte Carlo simulation framework with both academic problems and a 22nm 6T SRAM design.

5.4.1 Academic Test Case: Cantilever problem

The first test case is the cantilever problem (Equation 5.11). We set the failure boundary as \( f(x) > 18.3 \). With 10 billion Monte Carlo samples, the failure rate is estimated as \( p_{\text{fail}} = 7 \times 10^{-7} \) with a relative error of 1.24%. We regard this as the golden estimate.

We plot the PDF and \( f(x) \) values along with the sample distributions of iteration 1, 2, 4, and 8 in Figure 5.2. From the sampling iteration, it is apparent that with the guidance of the surrogate model, the newly ordered samples gradually move towards the failure boundary. This indeed proves the desirable effectiveness of surrogate model guided Monte Carlo sampling.

The iterative sampling process is automatically terminated after 10 iterations without acquiring any new failed sample (we deem this the convergence signal) or when the algorithm exceeds the maximum allowed iteration number.

In this example, we fix the random seed and tested the number of failure samples. With brute-force testing, we identify there are totally 65 failure samples among 100 million ones. The convergence plot shows the comparisons among the FFX, the RF, and the GB model in Figure 5.3. We see that the ensemble models and the FFX model eventually find all the failed samples, but it takes sample evaluations for the FFX model-based method to converge, compared to the random forest-based method, the gradient boosting model's
Figure 5.2 Sample distribution of iteration 1, 2, 4, and 8
performance is between the other two. This demonstrates our claim about the benefit of ensemble learning models.

The TOT comparison is shown in Table 5.5. We notice from Table 5.2 that FFX model is faster in each iteration compared with the other two, and gradient boosting model is the slowest. However, in actual circuit yield analysis, the most dominating part is always the circuit simulation. Therefore, we use the number of sample evaluations as our primary considering factor. For the convergence speed, the ensemble models spend less time to find all the failed samples.
Table 5.5 Cantilever performance comparison

<table>
<thead>
<tr>
<th>Model Type</th>
<th>FFX</th>
<th>GB</th>
<th>RF</th>
</tr>
</thead>
<tbody>
<tr>
<td>Failures Found</td>
<td>65</td>
<td>65</td>
<td>65</td>
</tr>
<tr>
<td>Converged Iteration</td>
<td>10</td>
<td>6</td>
<td>4</td>
</tr>
<tr>
<td>Sample Number for Convergence</td>
<td>547</td>
<td>345</td>
<td>248</td>
</tr>
</tbody>
</table>

With 65 failure cases among 100 million, the point estimate of the failure probability is:

\[ \hat{P}_f = \frac{65}{10^8} = 6.5 \times 10^{-7}. \]  

(5.12)

Using the Wilson’s method, we have the 95% confidence interval associated with this estimation: \((5.1 \times 10^{-7}, 8.2 \times 10^{-7})\). We see that the true failure rate \(7 \times 10^{-7}\) is within this estimated interval, therefore the estimation correct. Note that the confidence interval is a bit wide, this can be improved by increasing the pre-assigned Monte Carlo sample number, which is currently \(10^8\).

5.4.2 6T SRAM Test Case

In this section, we test the SMMC package on a 22nm 6T-SRAM case. The schematic is shown in Figure 5.4.

There are totally 6 random variables, \(i.e.\) the threshold voltages of the 6 transistors in the SRAM cell. We summarize their distributions in Table 5.6. The design specification requires that the static noise margin (SNM) should be greater than 0.27. As a golden reference, we ran a brute-force Monte Carlo simulation using \(10^9\) samples and calculated the failure rate as \(p_f = 4 \times 10^{-7}\).

The convergence plot is shown in Figure 5.5. Again, we notice that ensemble models

123
Figure 5.4 SRAM schematics

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Distribution</th>
<th>mean</th>
<th>std</th>
</tr>
</thead>
<tbody>
<tr>
<td>$V_{t1}$</td>
<td>normal</td>
<td>0</td>
<td>$25 \times 10^{-3}$</td>
</tr>
<tr>
<td>$V_{t2}$</td>
<td>normal</td>
<td>0</td>
<td>$25 \times 10^{-3}$</td>
</tr>
<tr>
<td>$V_{t3}$</td>
<td>normal</td>
<td>0</td>
<td>$35 \times 10^{-3}$</td>
</tr>
<tr>
<td>$V_{t4}$</td>
<td>normal</td>
<td>0</td>
<td>$35 \times 10^{-3}$</td>
</tr>
<tr>
<td>$V_{t5}$</td>
<td>normal</td>
<td>0</td>
<td>$35 \times 10^{-3}$</td>
</tr>
<tr>
<td>$V_{t6}$</td>
<td>normal</td>
<td>0</td>
<td>$35 \times 10^{-3}$</td>
</tr>
</tbody>
</table>
converges faster than the FFX model. It only takes around 1500 simulations to find all failure cases using ensemble models, while FFX model takes more than 3000.

Totally, there are 46 failure cases captured among the 100 million prepared Monte Carlo samples, and the failure rate of the 6T SRAM cell is estimated as:

\[ \hat{P}_f = \frac{46}{10^8} = 4.6 \times 10^{-7}. \]  

(5.13)

with the 95% Wilson confidence interval identified as (3.45\times10^{-7}, 6.14\times10^{-7}), which contains the golden reference failure rate of 4 \times 10^{-7}. 

\textbf{Figure 5.5} Convergence plot of the 6T-SRAM example
5.5 Conclusion

In this chapter, we discussed the surrogate model-based yield estimation technique and our improvement of the HSMC algorithm using ensemble models. Experiment results on the academic cantilever example and the 6T SRAM case indicate that ensemble models will effectively improve the algorithm's convergence time by a factor of up to 2. In addition, ensemble models also possess the capability of indicating prediction confidence level, this trait makes them suitable to be incorporated into a unified circuit IP modeling and optimization framework beyond the yield analysis application. We will leave this unified framework as a part of our next steps.
In this work, we first reviewed the bottleneck of EDA in the face of growing design complexity, with a focus on the simulation-based design approaches, and then we proposed our solution using machine learning and surrogate models. A brief review of the modeling techniques and algorithms is also presented to establish the foundation for our discussions. For the design applications, in Chapter 3, we demonstrated the use of surrogate models for fast design exploration and physical design flow configuration. An automated analog design optimization and IP reuse framework based on the Bayesian optimization algorithm
follows in Chapter 4, with which we redesigned two 77GHz analog IP blocks in the IBM 8XP technology and reused them in IBM 9HP. The final designs meet the tape-out standards with a minimal human labor involved. In the last part, we improved the HSMC algorithm using ensemble models and improved the convergence rate by a factor of up to 2. The efficacy of our improved algorithm is verified with an industrial 22nm 6T SRAM example.

We believe that machine learning will become a part of the EDA’s future. The works in this dissertation as well as the Center for Advanced Electronics through Machine Learning (CAEML), in which we are affiliated to, is only a beginning of this new era, as there are many potential applications and realms in EDA awaiting to be explored. At the end of this dissertation, we would like to discuss and identify some possible future steps.

1. **High-dimensional optimization:**

   As we demonstrated in Chapter 4, when optimizing a circuit with many design parameters, a screening step is performed not only to give designers feedback on the impacts of each parameter, but also make the optimization task easier. A common trick as it is in engineering designs, when various design parameters contribute equally to the final performance (for instance, a SRAM array), the Bayesian optimization engine is challenged from two aspects. Firstly, the difficulty in training the Gaussian processes or Student-\(t\) processes, as the training and predicting cost is \(O(n^3)\) where \(n\) is the dimensionality; and secondly, the difficulty in optimizing the acquisition function also becomes nontrivial. One possible step is to use additive Gaussian processes\[64\] that decompose the higher dimensional model into an additive set of independent lower dimensional ones. Note this idea is rather similar to the HDMR method, but their major difference is the assumption of statistical independence. The enforced
independence assumption help to "regularize" the GP model and prevents overfitting issue, as demonstrated in [64].

2. **High-dimensional yield analysis:**

   Although our improved SMMC algorithm uses ensemble models that can deal with high dimensionality, one slight drawback is the requirement of on-the-fly memory utilization. We optimized this issue by saving the random seed and only recreate the samples when needed. Still, this recreation step can gradually become costly when dimension increases in the rare-event scenario. As dimensionality does not influence the performance of crude Monte Carlo simulation, one way to preserve this nice trait while effectively handling rare-event yield analysis is through subset simulation. [125] has discussed the efficacy of subset simulation in a 300-dimensional case. However, as subset simulation relies on Markov Chain Monte Carlo (MCMC) to estimate conditional probabilities, the sample utilization is rather high. A potential solution is to use surrogate models in the subset simulation flow, and estimate the conditional probability either statistically [9] or by running the MCMC using this model. For the latter case, a predictive model that handles high dimensionality should be considered, such as the ensemble model shown in Chapter 5.

3. **Functional verification:**

   Another potential area of machine learning is in functional verification. Based on standard industry estimates, functional verifications take approximately 70% of the total efforts on the projects. Machine learning can help in at least three areas:

   a) **Test filtering:**
Many legacy tests merge into newly developed ones. Since there is an enormous amount of test cases and each test is expensive to execute, there is a need to pick the cases which cover unique scenarios, and identify duplicated cases or the cases that provide similar coverage although seemingly different. This is a typical setting for NLP-like classification. Similar works include [17, 47]. As an improvement, we can incorporate deep learning methods to automatically build features, as opposed to using domain knowledge.

b) Regression test and job configuration:
In this task, we need to identify when to run a test, what test to run, and how to rank and set their priority. This is indeed an optimization problem that we can solve using the Bayesian optimization technique.

c) Coverage closure:
This task required generating test cases or constraints/insertions to reach desired coverage levels and are usually done either manually or through constrained random tests. There have been some works in coverage directed test generation (CDTG) and insertion generation using statistical learning, such as [32, 121, 129], all of which provides a ground for us to further explore.

4. Design for Manufacturability (DFM):
At the layout level, many details left out from higher level abstractions need to be reconsidered to achieve manufacturable designs, such as hotspot detection, electromigration (EM), optical proximity correction (OPC), and layout parasitics pre-extraction. Here, we try to identify the potential research areas in DFM:
a) Hotspot detection using deep neural networks:
Machine learning based hotspot detection are discussed in [15, 26], for example. The idea is to extract features from layouts and use classifiers to tell if some particular layout patterns form potential hotspots. One improvement could be using convolutional dimension reduction technique (convolutional RBM, for instance) to preserve spatial information and automatically generate features, or even directly apply convolutional neural networks. Both will make the framework more generally applicable and capable to be automated, as less domain knowledge will be involved.

b) Optical Proximity Correction (OPC):
OPC deals with pattern mismatch from designed layout to manufactured layout during the lithographic process. This is typically done with rule-based approach, but recently, model-based approach, including machine learning, proves to be a better solution. The core problem is a geometric mapping from the designed layout shape to the shape after lithography. Deep learning is known for its capability of handling spatial and geometric pattern recognition, thus can be a good candidate if sufficient training example is available.

c) Parasitic pre-extraction:
In full custom IC design practice, the parasitics are extracted after the layout is designed (usually by hand) and designers need to reiterate the process if the layout fails the sanity check. We could use machine learning models to predict the impact of a particular layout pattern and use it to either provide early feedback to the designers or automate the layout generation process. For the former use, Cadence had an Electrically Aware Design (EAD) tool in the Virtuoso suite that lay the foundation for
this idea. Based on this pre-extraction tool, we could further utilize reinforcement learning approach to strategically guide the layout steps aiming at maximizing the final expected return and automatically (or semi-automatically) generate the layout and avoid design reiteration.
BIBLIOGRAPHY


[104] Ng, A. “CS229 Lecture notes” ()


[118] Shah, A. et al. “Student-t Processes as Alternatives to Gaussian Processes.”


APPENDIX
A.1 Surrogate Modeling Flow Code

A.1.1 The Parallel Flowchart

A.1.2 Code: The central script

This is the central perl script that controls all servers, including job distribution and result collection. Note that the absolute path to the postoffice directory is replaced with
"./postoffice".

#!/usr/local/bin/perl

# This perl script is for multi-server simulation,
# on olympia,thera and 5 grendel servers.
# We are running the PR problem, using the fastest netlist.

system('rm -f ./postoffice/*');
for(my $m=1; $m<22; $m++){
    # 21 simulations in parallel
    system("rm -f ./sim$m/sp21*");
    system("rm -f ./sim$m/sptri*");
    system("rm -f -r ./sim$m/*.lsn");
    system("rm -f -r ./sim$m/*.log");
    system("rm -f -r ./sim$m/*.out");
    system("rm -f -r ./sim$m/*.p1");
    system("rm -f -r ./sim$m/*.p2");
    system("rm -f -r ./sim$m/*.raw");
    system("rm -f -r ./sim$m/*.txt");
    system("rm -f -r ./sim$m/*.ba0");
    system("rm -f -r ./sim$m/*.bo0");
    system("rm -f -r ./sim$m/.api*");
    system("rm -f -r ./sim$m/.cadence");
    system("rm -f -r ./sim$m/.raw*");
}
system("rm -f -r ./sim$m/.rel*");
system("rm -f -r ./sim$m/.ring*");
system("rm -f -r ./sim$m/.*.dpl");
system("rm -f -r ./sim$m/.*.ic");
system("rm -f -r ./sim$m/.*.fc");
system("rm -f -r ./sim$m/newfile");
system("rm -f -r ./sim$m/*.ahdlSimDB");
}

while (1) {
    if (-e "./postoffice/simulate_these") {
        sleep(2);
        open(F,"./postoffice/simulate_these") || die("Can’t open!
");
        @samples=<F>;
        close F;
        chomp(@samples);
        $size=scalar @samples;
        for (my $i=1; $i<=$size; $i++){
            ($PRI[$i],$PRQ[$i],$IDrain[$i],$QDrain[$i],
            $INgate[$i],$IPgate[$i],$QNgate[$i],$QPgate[$i],
            $age[$i]) = split /\s+/, samples[$i-1];
        }
        printf("Got commands to simulate: ".
    }
}
for (my $i=1; $i<=$size; $i++){
    printf("PRI[\$i],PRQ[\$i],IDrain[\$i],\n".
            "$QDrain[\$i],INgate[\$i],IPgate[\$i],\n".
            "$QNgate[\$i],QPgate[\$i],\$age[\$i]\n"");
}
sleep(2);
system('rm -f ./postoffice/*');
for(my $m=1; $m<22; $m++){  
    system("rm -f ./sim$m/sp21*");  
    system("rm -f ./sim$m/sptri*");
}

# According to the # of samples,  
# we decide how many netlists to modify.
for (my $i=1; $i<=$size; $i++){
    my $in="./sim$i/PR_fastest.scs";  
    my $out="./sim$i/tmp_nlist.scs";
    unless(open IN, "<$in"){
        die "cant open: $!";}
    unless(open OUT, ">$out"){
        die "cant open: $!";}
while(<IN>){
    s/PR_I_set=\d+.\d+/PR_I_set=$PRI[$i]/;
    s/PR_Q_set=\d+.\d+/PR_Q_set=$PRQ[$i]/;
    s/Idrain=\d+.\d+/Idrain=$IDrain[$i]/;
    s/Qdrain=\d+.\d+/Qdrain=$QDrain[$i]/;
    s/IN_gate=\d+.\d+/IN_gate=$INgate[$i]/;
    s/IP_gate=\d+.\d+/IP_gate=$IPgate[$i]/;
    s/QN_gate=\d+.\d+/QN_gate=$QNgate[$i]/;
    s/QP_gate=\d+.\d+/QP_gate=$QPgate[$i]/;
    s/.age \d+.\d+/.age $age[$i]/;
    s/ sigma=.?\d+.+/ sigma=$sigma[$i]/gi;
    print OUT $_;
}

system("cp ./sim$i/tmp_nlist.scs ./sim$i/PR_fastest.scs");
system("touch ./sim$i/newfile");
printf "Now we have put newfile in sim$i \n";
}

# Wait for all $size simulations to finish
for (my $j=1; $j<=$size; $j++){
    $done[$j-1] = "./sim$j/sp21_out";
}
$doneflag = 0;

151
while($doneflag==0){
    $doneflag = 1;
    foreach $k (@done){
        $doneflag = 0 unless -e $k;
    }
    sleep(5);
}

undef @done;
printf "Finished $size simulations~!! \n";

# Result handling.
for (my $i=1;$i<=$size;$i++){
    system("cp ./sim$i/sp21_out ./postoffice/newdata_sp$i");
    if ($i==1){ next;}
    else {
        system("cat ./postoffice/newdata_sp$i >>".

            " ./postoffice/newdata_sp1");
    }
}

system("chmod 777 ./postoffice/*");
system("cp ./postoffice/newdata_sp1 ./postoffice/newdata_sp");
system("chmod 777 ./postoffice/*");
}
else {
    print "SP:Pausing for 5 seconds!!!\n";
    sleep 5;
}

__END__

A.2 Bayesian Optimization

A.2.1 Code: BO API example

This code is an API connecting the current simulator to the BayesOpt library[90]. Demonstrated here is the power amplifier test case.

#define _USE_MATH_DEFINES
#include "testfunctions.hpp"
#include <ctime>
#include <iostream>
#include <fstream>
#include "param_loader.hpp"
#include <cmath>
#include <algorithm>
#include <boost/math/constants/constants.hpp>
#include <boost/numeric/ublas/assignment.hpp>
#include "bayesopt/bayesopt.hpp"
#include "specialtypes.hpp"

using namespace std;

class Optimize_PA_new: public bayesopt::ContinuousModel {
public:
    Optimize_PA_new(bayesopt::Parameters param):
        ContinuousModel(7, param) {

double evaluateSample( const vectord& xin ) {
    if (xin.size() != 7) {
        cout << "WARNING: This only works for 7-D inputs." << endl
        << "WARNING: Using only first 7 components." << endl;
    }

    string params = "";
    for (int i = 0; i < xin.size(); i++) {
        string s = boost::lexical_cast<std::string>(xin[i]);
        params += (s + " ");
    }

    string command = "python ",
    string root_path = "/home/wqi/10_optimize_PA_9HP_1/",
    string script_name = "mod_netlist.py ",
    command += (root_path + script_name + params);
    // start simulation

system(command.c_str());

// get result
fstream resultFile("./simulation_result", ios_base::in);
float result;
resultFile >> result;

system("rm -rf ./simulation_result");
    return 0 - result;  // maximization problem 
};

bool checkReachability(const vectord &query) {
    return true;
};

int main(int nargs, char *args[]) {
    bayesopt::Parameters par;
    if(nargs > 1) {
        if(!bayesopt::utils::ParamLoader::load(args[1], par)){
            cout << "ERROR: provided file " << args[1] << " does not exist" << endl;
            return -1;
        }
    } else {
        par = initialize_parameters_to_default();
        par.n_iterations = 400;
    }
par.n_iter_relearn = 1;
par.n_init_samples = 20;
par.noise = 1e-10;
par.random_seed = 0;
par.verbose_level = 1;
par.crit_name = "cEI";
par.load_save_flag = 2;
par.surr_name = "sStudentTProcessNIG";
par.l_type = L_MCMC;

bayesopt::utils::ParamLoader::save("bo_PA_9HP.txt", par);
}

Optimize_PA_new PA(par);

vectord lowerBounds(7); vectord upperBounds(7);
lowerBounds(0) = 35; upperBounds(0) = 350;
lowerBounds(1) = 35; upperBounds(1) = 350;
lowerBounds(2) = 4; upperBounds(2) = 24;
lowerBounds(3) = 35; upperBounds(3) = 350;
lowerBounds(4) = 35; upperBounds(4) = 350;
lowerBounds(5) = 35; upperBounds(5) = 350;
lowerBounds(6) = 4; upperBounds(6) = 24;
PA.setBoundingBox(lowerBounds, upperBounds);
ofstream timelog;
timelog.open("time_PA_9HP_morris2.log");
clock_t curr_t;
clock_t prev_t = clock();
PA.initializeOptimization();
for (size_t ii = 0; ii < par.n_iterations; ++ii) {
    PA.stepOptimization();
curr_t = clock();
timelog << ii << ","
    << static_cast<double>(curr_t - prev_t) / CLOCKS_PER_SEC
    << std::endl;
    prev_t = curr_t;
}
timelog.close();
return 0;
}

A.2.2 Code: Simulator script

This is the "mod_netlist.py" example used by the code above. In this Python script, we modify the netlist per the request of the optimization engine, simulate it, and parse the result. There is also an simulation timeout handling routine to prevent the simulation process from being stuck due to technology model convergence issue.

#!/usr/bin/python
import os, sys, re
from numpy import *
import subprocess as sub
import threading

# This class is here to handle long simulations
# we will wait for "timeout" seconds, if it still haven’t finished,
# then kill it and resubmit

class run_cmd(threading.Thread):
    def __init__(self, cmd, timeout):
        threading.Thread.__init__(self)
        self.cmd = cmd
        self.timeout = timeout

    def run(self):
        #print self.cmd
        self.p = sub.Popen(self.cmd, shell=True)
        self.p.wait()

    def Run(self):
        self.start()
        self.join(self.timeout)

        if self.is_alive():
            print "- Current simulation is still running!"
print "- Killing it!"
self.p.kill()
o.s.system("killall virtuoso")
o.s.system("killall spectre")
self.join()
return 0

return 1

currentDir = os.getcwd()
o.s.chdir(sys.path[0])

#########################################################################
# modify the netlist according to arguments
#########################################################################

if len(sys.argv) != 9:
    print "Wrong number of inputs!!"
    exit()
nl_mod = open("./netlist/netlist", 'r')
nl_mod_tmp = open("./netlist/netlist_tmp", 'w')
replace_param = """parameters MLPARAMa=%fu MLPARAMb=%fu
MLPARAMc=%fu MLPARAMd=%fu MLPARAMe=%fu
MLPARAMf=%fu MLPARAMg=%fu """ %
(float(sys.argv[1]), float(sys.argv[2]),
  float(sys.argv[3]), float(sys.argv[4]),
  float(sys.argv[5]), float(sys.argv[6]),
  float(sys.argv[7]))

lines = nl_mod.readlines()
lines[0] = replace_param
for line in lines:
    nl_mod_tmp.write(line)
nl_mod.close()
nl_mod_tmp.close()
o.s.system("mv ./netlist/netlist_tmp ./netlist/netlist")

cmd = "./run_commands > /dev/null"
T0 = 900
max_failure = 2
failure = 0
success = run_cmd(cmd, T0).Run()
failure += (1 - success)

while not success and failure < max_failure:
    # re-run failed simulation
    print "previous simulation failed, simulate again..."
print "number of failure: ", failure
sys.stdout.flush()
success = run_cmd(cmd, TO).Run()
failure += (1 - success)

if failure >= max_failure:
    print "Simulation failed after %d tries, giving up..." % max_failure
    failed_sampleFile = open("./failed_simulated_samples", 'a')
    inputs = ",".join(str(float(i)) for i in sys.argv[1:])
    failed_sampleFile.write( inputs + "," + str(0) + "\n"");
    failed_sampleFile.close()
    target1 = 0.0
    target2 = 99.0
else:
    testdata1 = genfromtxt("./myOcnprintFile01",
                           dtype='float',
                           skip_header=4)
    target1 = amin(testdata1[:,1])
    if target1 > 40: # rule out wrong results
        target1 = 11.11
    testdata2 = genfromtxt("./myOcnprintFile02",
                           dtype='float',
                           skip_header=3)[1:, 1:]

161
target2 = amax([std(testdata2[i, :])
    for i in range(testdata2.shape[0])])

os.system("rm -f myOcnprintFile*")
os.chdir(currentDir);
inputs = ",".join(str(float(i)) for i in sys.argv[1:])
sampleFile = open("./simulated_samples", 'a')
sampleFile.write( inputs + ","
    + str(target1) + ","
    + str(target2) + "\n");
sampleFile.close()

resultFile = open("./simulation_result", 'w')
resultFile.write(str(target1))
resultFile.close()

A.3 SMMC

A.3.1 Code: SMMC API example

We present the API for using the SMMC package described in Chapter 5. This API is for the cantilever problem.

```python
import smmc

def f(X):
    # Structural reliability test example
```
# Ling Li, et al. BSS: a kriging-based subset simulation algorithm
# for the estimation of small probabilities of failure
# Failure: f(x) >= 17.8

    return 18.46154 - 7.476923 * 10**10 * X[:,0] / (X[:,1])**3

dists = ['normal','normal']
params = [{'name': 'x1', 'mean': 0.001, 'std': 0.0002},
          {'name': 'x1', 'mean': 250, 'std': 37.5}]

p = smmc.Problem(name = "test_func", func = f,
                  dists = dists, params = params,
                  seed=1)

smmc.SimulateSMMC(p, 17.5, desired_sample_number=10000000,
                  n_iter = 10, verbose=True, plot = True,
                  n_init = 100, n_additional = 50)