Titre: Title:	Statistical Methods for Efficient Digital Electronic Hardware Design
Auteur: Author:	Seyed Alireza Ghaffari
Date:	2023
Type:	Mémoire ou thèse / Dissertation or Thesis
Référence: Citation:	Ghaffari, S. A. (2023). Statistical Methods for Efficient Digital Electronic Hardware Design [Ph.D. thesis, Polytechnique Montréal]. PolyPublie. https://publications.polymtl.ca/54400/

URL de PolyPublie: PolyPublie URL:	https://publications.polymtl.ca/54400/
Directeurs de recherche: Advisors:	Yvon Savaria
Programme: Program:	Génie électrique

POLYTECHNIQUE MONTRÉAL
affiliée à l'Université de Montréal

Statistical Methods for Efficient Digital Electronic Hardware Design

SEYED ALIREZA GHAFFARI
Département de génie électrique

Thèse présentée en vue de l'obtention du diplôme de *Philosophiæ Doctor*
Génie électrique

Juin 2023

POLYTECHNIQUE MONTRÉAL
affiliée à l'Université de Montréal

Cette thèse intitulée :

Statistical Methods for Efficient Digital Electronic Hardware Design

présentée par **Seyed Alireza GHAFFARI**
en vue de l'obtention du diplôme de *Philosophiæ Doctor*
a été dûment acceptée par le jury d'examen constitué de :

Pierre LANGLOIS, président
Yvon SAVARIA, membre et directeur de recherche
Tarek OULD-BACHIR, membre
Mounir BOUKADOUM, membre externe

DEDICATION

To my mother,
To the loving memory of my father,
. . .

ACKNOWLEDGEMENTS

I would like to express my heartfelt gratitude to everyone who has supported me throughout my PhD journey.

I would like to thank my supervisor, Professor Yvon Savaria, for his invaluable guidance, expertise, and support. His feedback and encouragement have been instrumental in shaping my research skills and preparing me for the challenges of a research career.

I am sincerely grateful to Professor Masoud Asgharian, for his mentorship, guidance, and support during my stay at McGill University. His diversity of perspectives, encouragement, and patience enabled me to navigate the challenges and opportunities that arose.

I am also deeply grateful to my thesis committee members, Professor Pierre Langlois, Professor Mounir Boukadoum and Professor Tarek Ould-Bachir, for agreeing to review my thesis and for insightful comments, critical feedback, and constructive suggestions. Their expertise has enriched this thesis.

I would like to sincerely thank my friend, Dr. Vahid Partovi Nia, who generously shared his time and insights with me.

I am happy to have friends who dragged me to the finish line; Yasaman, Melanie, Yavar, Andishe, Hamed, and Vanessa. Their unwavering support, understanding, and encouragement have been a source of inspiration and motivation throughout this journey and for the many ways in which they have enriched my life.

My PhD journey would not have been complete without the support and encouragement of my family and loved ones. A special acknowledgment should be made to my mother for her patience and love while completing this degree. Words cannot express my gratitude and love for her. I could not have done this without the support of my sisters; I cannot imagine my life without them. I would like to acknowledge my beloved father and dedicate this thesis to the memories he hacked in my heart; I can never be thankful enough for him and his great personality that influenced and will influence my entire life.

During my time at Poly, there were moments that made me think deeper about myself, moments of disillusionment and clarity. I think these moments have an unwritten thesis of their own, more important than this one.

RÉSUMÉ

Avec la complexité croissante de nombreuses nouvelles applications qui servent notre société moderne, il est essentiel de concevoir des plates-formes informatiques efficaces. Cependant, la conception de matériel efficace est un problème complexe à objectifs multiples et qui est influencée par de nombreux paramètres. Étant donné le nombre élevé de paramètres et objectifs qui sont impliqués dans ce processus de conception, la synthèse de toutes les combinaisons possibles n'est pas une méthode viable pour trouver la solution optimale. Ainsi, les chercheurs souhaitent trouver des methodes plus efficaces pour modéliser le matériel et effectuer une exploration de l'espace de conception. Cette recherche vise à optimiser le processus de conception matérielle d'algorithmes à calcul intensif utilisant l'intelligence artificielle. Pour atteindre cet objectif, diverses techniques d'intelligence artificielle, telles que l'apprentissage actif, l'apprentissage par renforcement et l'apprentissage automatique statistique, sont envisagées pour résoudre ce problème difficile.

Cette thèse propose d'utiliser des algorithmes de recherche méta-heuristiques intelligents tels que l'optimisation Grey Wolf (GWO) en conjonction avec l'optimisation Bayésienne (BO) pour effectuer l'exploration de l'espace de conception matérielle. Nous montrons que nous pouvons réduire davantage l'effort de conception en utilisant un modèle de substitution créé sur la base de notre méthode hybride GWO-BO proposée. Le modèle de substitution est une abstraction utile pour détecter les interdépendances fonctionnelles et physiques dans le système afin de prédire avec précision ses performances (par exemple, le débit ou la latence). Nous évaluons notre méthodologie et montrons qu'elle peut produire des résultats compétitifs pour trouver les meilleurs paramètres de conception qui maximisent les performances du système. De plus, nous proposons une approche d'apprentissage actif basée sur un modèle pour réaliser la modélisation des performances du matériel. Notre méthode proposée utilise des modèles Bayésiens pour caractériser divers aspects des performances matérielles. Nous utilisons également des techniques d'apprentissage par transfert et d'amorçage de régression Gaussienne en conjonction avec un apprentissage actif pour créer des modèles plus précis. Notre méthode de modélisation statistique proposée fournit des modèles matérielles suffisamment précis pour effectuer simultanément l'exploration de l'espace de conception et la prédiction des performances. Nous utilisons notre méthode proposée pour effectuer l'exploration de l'espace de conception et la prédiction des performances pour diverses configurations matérielles, telles que la conception de micro-architecture et les noyaux OpenCL pour les FPGAs. Nos expériences montrent que le nombre d'échantillons nécessaires pour créer des modèles de performance diminue considérablement tout en maintenant le pouvoir

prédictif de nos modèles statistiques proposés. Par exemple, la méthode proposée nécessite 65 % d'échantillons en moins pour créer le modèle de prédiction des performances. De plus, Dans le cadre de l'exploration de l'espace de conception, notre méthode proposée peut trouver les meilleurs paramètres de conception en explorant aussi peu que 50 échantillons.

ABSTRACT

With the rising complexity of numerous novel applications that serve our modern society comes the essential need to design efficient computing platforms. Designing efficient hardware is, however, a complex multi-objective problem that deals with multiple parameters and their interactions. Since many parameters and objectives are involved in hardware design, synthesizing all possible combinations is not a feasible method to find the optimal solution. Thus, researchers are interested in finding more intelligent approaches to model the hardware and perform design space exploration. This research aims to optimize the hardware design procedure of compute-intensive algorithms using artificial intelligence. To achieve this goal, various artificial intelligence techniques, such as active learning, reinforcement learning, and statistical machine learning, are envisioned to solve this challenging problem.

We propose using intelligent meta-heuristic search algorithms such as Grey Wolf Optimization (GWO) in conjunction with Bayesian Optimization (BO) to perform hardware design space exploration. We show that we can further reduce the design effort using a surrogate model created based on our proposed hybrid GWO-BO method. The surrogate model is a useful abstraction to detect functional and physical inter-dependencies in the system to predict its performance (e.g. throughput or latency) accurately. We evaluate our methodology and show that it can produce competitive results to find the best design parameters that maximize the system's performance. Additionally, we propose a model-based active learning approach to accomplish performance modeling of hardware. Our proposed method uses Bayesian models to characterize various aspects of hardware performance. We also use transfer learning and Gaussian regression bootstrapping techniques in conjunction with active learning to create more accurate models. Our proposed statistical modeling method provides hardware models that are sufficiently accurate to perform design space exploration and performance prediction simultaneously. We use our proposed method to perform design space exploration and performance prediction for various hardware setups, such as micro-architecture design and OpenCL kernels for FPGA targets. Our experiments show that the number of samples required to create performance models significantly reduces while maintaining the predictive power of our proposed statistical models. For instance, the proposed method needs 65% fewer samples to create the model in the performance prediction setting. Moreover, our proposed method can find the best design parameter settings in the design space exploration setting by exploring as few as 50 samples.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF SYMBOLS AND ABBREVIATIONS

A2C	Advantage Actor Critic
ADRS	Average Distance to Reference Set
ALM	Adaptive Logic Module
BBO	Black Box Optimization
BF-DSE	Brute Force Design Space Exploration
BFS	Breadth-First Search
BIPS	Billion Instructions per Second
BO	Bayesian Optimization
CAD	Computer Aided Design
CNN	Convolutional Neural Network
CONV	Convolution
CPU	Central Processing Unit
DCT	Discrete Cosine Transform
DSE	Design Space Exploration
DSP	Digital Signal Processing
FPGA	Field Programmable Gate Array
FIFO	First In First Out
FIR	Finite-Impulse Response
GA	Genetic Algorithm
GD	Gradient Descent
GEMM	General Matrix Multiplication
GPU	Graphics Processing Unit
GWO	Grey Wolf Optimization
HC	Hill Climbing
HC-DSE	Hill Climbing Design Space Exploration
HLS	High-Level Synthesis
I/O	Input/Output
IoT	Internet of Things
jbb	Java Business Benchmark
LASSO	Least Absolute Shrinkage and Selection Operator
LSTM	Long Short Term Memory
LUT	Lookup Table
MSE	Mean Squared Error

ONNX	Open Neural Network eXchange format
PLL	Phase-Locked Loop
RAM	Random Access Memory
RL	Reinforcement Learning
RL-DSE	Reinforcement Learning Design Space Exploration
RNN	Recurrent Neural Network
RTL	Register Transfer Level
SGD	Stochastic Gradient Descent
SIMD	Single Instruction Multiple Data
SIMT	Single Instruction Multiple Thread
SoC	System-on-Chip
SPMV	Sparse Matrix-Vector Multiplication
TL	Transfer Learning
VHDL	Very High-Speed Integrated Circuit Hardware Description Language

CHAPTER 1 INTRODUCTION

1.1 Motivation

There are numerous novel algorithms that serve our modern society. These algorithms include but are not limited to deep learning, augmented reality, virtual reality, and advanced 5G/6G communications. Over the past decade, human daily life is becoming more dependent on such compute-intensive algorithms. Thus, there is a need to design efficient computing platforms that not only provide the required computational power but also are energy efficient. Designing efficient hardware also has a socio-economical effect on human life. The annual electricity bill of an average-sized data center is approximately 3 million U.S. dollars [1]. Most of this electrical energy is consumed to cool down the computing servers. Designing efficient computing clusters that generate less heat for the same computational power is very interesting for the hardware design community. Efficient computing clusters not only provide various affordable services to society but also have a smaller carbon footprint and, consequently fewer adverse effects on climate changes.

Designing efficient computing hardware is a complex multi-objective problem that deals with multiple parameters and their interactions. A large number of parameters makes design space exploration impossible with brute-force analysis and next to impossible for traditional search methods. Nowadays, the hardware design industry favors using intelligent hardware analysis before manufacturing. In many applications, multiple design parameters and their interactions must be considered before proceeding with the design synthesis. Having the complexity of the hardware design problem in mind, the need to have more effective approaches to explore the design options becomes extremely important. Therefore, one promising way to tackle this problem is the mathematical modeling of the computer hardware architecture. This thesis is directly related to statistical modeling of the hardware, where various methods such as *Bayesian optimization*, *reinforcement learning*, *regression analysis* and *Gaussian processes bootstraps* are used to model and optimize the hardware prior to the synthesis and manufacturing.

Ideally, this thesis aims to propose and explore means to construct a statistical *surrogate* model of the hardware. A statistical surrogate model is a useful abstraction that can reveal functional and physical inter-dependencies in the actual model. The surrogate model can be trained over a known set of training data and *predicts* the performance of the hardware for some unseen data and tasks. Using such surrogate models in the CAD tools can lead to faster time-to-market and lower development efforts for high-performance computing hardware.

1.2 Definition of the Main Concepts

Here, we elaborate on the general background concepts that are commonly used throughout this thesis. The first concept is the definition of a *surrogate model* in our application. The second and third concepts are *design space exploration* and *performance modeling* as two prominent problems in the field of hardware design research and the main focus of this thesis.

1.2.1 Surrogate Model

In many scientific problems, experimental results are required to evaluate design objectives and constraint functions as a function of design parameters. Sometimes the actual physical experiment is hard to design or is very time-consuming. In such situations, a mathematical model, notably a *surrogate model*, can mitigate the burden of physical experiments. A surrogate model is an abstract model that can be used to simulate or predict the outcome of an experiment.

Constructing a surrogate model consists of three main steps:

- **Sample selection:** The real-world samples are gathered to construct the surrogate model.
- **Modeling:** A mathematical model is created. In our proposed research, we use statistical methods to create our surrogate models.
- **Appraisal of the model:** The model is evaluated in terms of accuracy and predicting performance.

1.2.2 Design Space Exploration

Design Space Exploration (DSE) deals with the problem of finding the hardware design parameter set that maximizes a specific objective function. For example, in some applications, we want to know which design parameter set maximizes the throughput of a specific hardware design. Moreover, in the case of multi-objective optimization, DSE aims to find the dominant or Pareto optimal solutions. The model is accurate if

$$\arg\max(f_s) = \arg\max(O), \tag{1.1}$$

meaning that the surrogate function f_s can accurately predict the parameter set that maximizes the objective function. Also, note that here *argmax* is used for the objectives that need to be maximized, such as throughput. Likewise, the goal for the objective functions that are to be minimized, e.g. power consumption, is the same, except that arg max should be replaced by arg min.

1.2.3 Performance Modeling and Performance Prediction

In *performance modeling* problems, the goal is to accurately *predict* the hardware performance (or value of the objective function O) for a specific hardware design parameter set. Thus, performance modeling is essentially a regression problem, and the accuracy of the model can be measured with the *Mean Squared Error (MSE)* between the surrogate function f_s and the actual objective function O over N parameter sets $\mathbf{x}_{1:N} = \{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N\}$

$$\text{MSE} = \frac{1}{N}\sum_{i=1}^{N}\left(O(\mathbf{x}_i) - f_s(\mathbf{x}_i)\right)^2 = \frac{1}{N}\sum_{i=1}^{N}\varepsilon_i^2. \tag{1.2}$$

The model becomes more accurate if the errors ε_i are negligible. Hardware performance models are used for predicting the power consumption and throughput of a design.

1.3 Problem Statement

Efficient hardware design is essential for various industrial and scientific applications. More specifically, with the rise of new applications such as deep learning, augmented reality, and blockchain, there is a strong need to design computing platforms that provide the required computational power with a limited energy budget. The increasing number of design parameters in modern hardware makes brute-force analysis infeasible and urges designers to use more advanced hardware models to tackle this issue.

Here we quickly list the challenges that are associated with efficient hardware design and why the research on statistical surrogate models of hardware is essential to address these problems:

- Designing efficient digital hardware is a multi-objective optimization problem where the convexity and smoothness of the objectives are not guaranteed, which makes the problem more challenging.
- Understanding such multi-objective, non-linear non-convex problems is difficult for most human experts, and it is easy to miss optimal design solutions.

- Considering a large number of design parameters, performing a brute-force search to find the optimum design is not feasible because synthesizing complex hardware designs is very time-consuming.
- The interaction between design parameters and their related trade-offs are often unknown or differ from one design to another. This means that increasing one objective function may decrease the other objectives.
- Since design objectives are not smooth, and also most parameters are discrete (e.g. the memory delay, the number of floating-point arithmetic units, and clock frequencies), the objective function is not differentiable. Thus, using gradient-based methods, such as Stochastic Gradient Descent (SGD), to find the optimum parameter set is not possible.

All these challenges can be mitigated by proposing a proper hardware surrogate model. More specifically, a statistical surrogate model is made to emulate the performance of hardware for various design parameters and applications. A surrogate model has a closed mathematical form and is *not* time-consuming to create. Once a statistical surrogate model is created, it can be used effectively for trying various multi-objective optimization approaches. Moreover, a surrogate model can be used as an abstract model to study the interaction between design parameters and their related trade-offs and interdependencies.

On the other hand, other challenges may arise while using statistical surrogate models. A common problem is the model's accuracy, which directly determines its predictive power and reliability. Another challenge is sample selection while creating the statistical surrogate models. The performance of a statistical surrogate model is directly related to the samples that are used to construct the model.

In this thesis, we address all the aforementioned challenges. For instance, we propose a statistical modeling method that effectively models the hardware behavior. Moreover, we show that our proposed statistical models work with either convex or non-convex objectives. We explore sample selection strategies and their effect on the performance of our proposed statistical models. In addition, we use various statistical techniques to improve our proposed statistical modeling process.

1.4 Research Objectives

- General Objective:

  The general objective of this research is to optimize the hardware design procedure of compute-intensive algorithms using artificial intelligence. To achieve this goal, various

artificial intelligence techniques, such as active learning, reinforcement learning, and statistical machine learning, are envisioned to solve this challenging problem. Furthermore, novel statistical modeling approaches are proposed as means to obtain accurate and effective surrogate models for design analyses.

- Specific Objectives:

  1. The first objective is to automatically implement deep learning algorithms such as Convolutional Neural Networks (CNNs) on hardware. The deep learning algorithm must be defined using high-level deep learning frameworks such as Pytorch. To achieve this goal, an algorithm is needed to be able to extract information from the high-end deep learning framework and to perform design space exploration to fit the design under various hardware constraints. Furthermore, the secondary aim is to study Reinforcement Learning (RL) and Hill Climbing (HC) approaches for design space exploration of deep learning algorithms.
  2. The second objective is to investigate design space exploration methods further. Model-based optimization techniques, such as Bayesian Optimization, are suitable to achieve this goal. Since Bayesian models' accuracy are extremely sensitive to their sample selection, the sample selection methodology must be revisited and combined with known heuristic methods to provide samples with better quality for the Bayesian optimization method.
  3. The third objective is to provide a statistical surrogate modeling methodology that is not only capable of doing design space exploration but also is able to achieve performance modeling and performance prediction. Various statistical and machine-learning approaches can be combined to achieve this objective. These approaches are included but not limited to Active Learning, Bayesian optimization, Gaussian Regression Bootstrapping, and Transfer Learning.

1.5 Research Contributions

In the pursuit of the research objective mentioned in the previous section, the following journal publications propose a range of solutions for efficient hardware design problems. Also, note that all the proposed solutions use statistical and artificial intelligence approaches.

1. Ghaffari, A. and Savaria, Y., 2020. "CNN2Gate: An Implementation of Convolutional Neural Networks Inference on FPGAs with Automated Design Space Exploration." Electronics, 9(12), doi: 10.3390/electronics9122200.

2. Ghaffari, A. and Savaria, Y., 2021. "Efficient Design Space Exploration of OpenCL Kernels for FPGA Targets Using Black Box Optimization." IEEE Access, 9, doi: 10.1109/ACCESS.2021.3117560.

3. Ghaffari, A., Asgharian, M. and Savaria, Y., 2023. "Statistical Hardware Design With Multi-model Active Learning". Revised version submitted to IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems.

In Chapter 3, "CNN2Gate: An Implementation of Convolutional Neural Networks Inference on FPGAs with Automated Design Space Exploration", we proposed a framework for implementing CNNs on FPGA targets that (i) Performs generalized model analysis and extracts computational graph, weights, and biases, (ii) Performs automated high-level synthesis and (iii) Executes hardware-aware design space exploration using various techniques such as reinforcement learning and hill climbing.

In Chapter 4, "Efficient Design Space Exploration of OpenCL Kernels for FPGA Targets Using Black Box Optimization", made the following contributions: (i) Providing an automated framework to optimize OpenCL kernels in HLS, (ii) Proposing that Grey Wolf Optimization (GWO) is a competitive method to perform design space exploration, (iii) Building surrogate models that estimate the behavior of OpenCL kernels using a Bayesian Optimizer, (iv) Proposing a hybrid (GWO-BO) optimizer based on re-using the sample points explored by the GWO algorithm and feeding them to BO to create a surrogate model for HLS OpenCL kernels that systematically provides better results than both GWO and BO methods. To the best of our knowledge, this is the first time that GWO is proposed for hardware design space exploration. Furthermore, this is the first time that the explored variables of a meta-heuristic algorithm are re-used to build a more accurate surrogate model (i.e Hybrid GWO-BO method). The hybrid method enables benefiting from both algorithms without repeating expensive synthesis trials.

In Chapter 5, "Statistical Hardware Design With Multi-model Active Learning", we made the following contributions: (i) We propose a *model-based active learning* framework that generates statistical models of different aspects of hardware performance. (ii) We propose *Bayesian transfer learning* to effectively perform hardware design space exploration using prior hardware performance information. (iii) We use a *Gaussian regression bootstrapping* method to effectively estimate the behavior of the hardware from the limited samples acquired from real-world applications. To the best of our knowledge, this is the first time that model-based active learning has been used in conjunction with Bayesian transfer learning to provide accurate statistical hardware models for design space exploration. Furthermore, this

is the first time that Bayesian regression bootstrapping has been used to better estimate the behavior of specific hardware.

1.6 General Organization of the Thesis

This thesis consists of seven chapters that are organized as follows. Chapter 2 reviews the important background knowledge and related literature that are used in this thesis. Chapter 3 presents the first contribution as a journal paper that elaborates on the implementation of CNNs on FPGA targets and their design space exploration methods. Chapter 4 presents our second contribution as a journal paper and is dedicated to studying efficient design space exploration of the OpenCL kernels on FPGA targets. Chapter 5 presents our third contribution as a submitted journal paper. This chapter explores various statistical methods for the design of hardware and shows that these methods are efficient in both design space exploration and performance prediction applications. Chapter 6 presents a general discussion about the proposed contributions and emphasize the benefits and limitations of our work. Finally, Chapter 7 highlights the conclusions of this thesis by summarizing the important contributions and suggestions for future research directions.

CHAPTER 2 LITERATURE REVIEW

In the realm of hardware design optimization, a multitude of techniques and methodologies have been devised to address the intricate challenges presented by modern computing applications. This chapter starts with a comprehensive journey through the landscape of optimization approaches, delving into the breadth of methodologies that have been used to optimize various hardware designs. The review of heuristic methods, statistical methods, iterative methods, multi-objective optimization, and hardware optimization frameworks shed light into diverse avenues that researchers and engineers have explored to achieve hardware efficiency and performance.

In this chapter, Section 2.1 surveys heuristic methods, encompassing powerful algorithms like the Genetic Algorithm (GA) and the Grey Wolf Optimization (GWO). These methods epitomize the evolutionary nature of hardware design optimization, drawing inspiration from biological evolution and animal behavior to navigate complex design spaces.

Turning our attention to statistical methodologies, Section 2.2 explores approaches such as linear regression, Least Absolute Shrinkage and Selection Operator (LASSO), and random forest. Embracing the power of data-driven insights, these methods offer a systematic approach to modeling relationships between input variables and performance metrics.

Iterative strategies constitute the focal point of Section 2.3, where we delve into methodologies like Bayesian optimization, reinforcement learning, and active learning. These methods leverage iterative feedback loops to adapt and refine their strategies, gradually converging toward optimal solutions.

Section 2.4 embraces the nuances of multi-objective optimization, introducing the concept of Pareto frontiers and dominant solution sets. In scenarios where conflicting objectives necessitate trade-offs, multi-objective optimization strives to find solutions that lie on the Pareto Front, representing the optimal compromise between competing goals.

Section 2.5 surveys hardware optimization frameworks, which encapsulate a holistic approach to tackling the complex challenges of hardware design. Furthermore, Section 2.6 discusses the current limitations in the literature, which motivates our research.

2.1 Heuristic Methods

In the pursuit of hardware design optimization, heuristic methods have emerged as potent tools, exploiting the principles of biological evolution and animal behavior to navigate com-

plex design spaces. This section provides an overview of two prominent heuristic methods: Genetic Algorithm (GA) and Grey Wolf Optimization (GWO).

2.1.1 Genetic Algorithms (GAs)

Genetic Algorithms (GAs) leverage principles of natural selection to optimize solution spaces by employing selection, crossover, and mutation. GAs' versatility makes them applicable to a variety of hardware optimization challenges.

Bao and Watanabe [2] proposed a GA-based approach for circuit design optimization, accommodating complex architectures while considering circuit complexity, power, and time delay. Krishnan and Katkoori [3] presented a GA framework for datapath design space exploration, simultaneously addressing scheduling, allocation, and module selection tasks, meeting latency and area constraints. Bolchini et al. [4] introduce a multi-objective GA for reliable FPGA-based systems, combining hardware-software partitioning and design alternatives, enhancing system reliability. Srinivasan et al. [5] proposed an integrated approach to hardware-software partitioning and design space exploration, using GA for data-dominated designs in digital signal processing and image processing. Anderson and Khalid [6] presented SC Build, a tool employing GA to explore parameterized core designs for FPGAs, aiding in informed decisions and optimizing hardware platforms.

2.1.2 Grey Wolf Optimization (GWO)

Grey Wolf Optimization (GWO) draws inspiration from wolf pack behaviors to efficiently explore complex solution spaces. GWO employs hunting-inspired search operators to simulate the behaviors of alpha, beta, and delta wolves, allowing it to converge towards optimal solutions.

Marot et al. [7] introduced a multi-objective GWO framework for the joint hardware-software design of a radar system. This approach optimizes classification performance by simultaneously designing a radar sensor and image processing software.

Siddavaatam and Sedaghat [8] proposed GWO-DSE, a framework for architectural synthesis trade-offs. It leverages GWO's solution evaluation approach and fitness function to enhance power-performance-area trade-offs, outperforming existing methods.

Prakasam et al. [9] applied GWO to constrained hardware-software co-design partitioning, showcasing its efficacy in optimizing this crucial step in embedded system design.

Khah et al. [10] utilized multi-objective GWO to optimize a high-speed one-bit full adder

circuit, improving power consumption and delay time through MOSFET channel width optimization.

In summary, Grey Wolf Optimization (GWO) adapts wolf behaviors to efficiently explore complex solution spaces. Its applications span joint hardware-software design, architectural synthesis trade-offs, hardware-software co-design partitioning, and hardware circuit optimization. This method has been extensively used in Chapter 4.

2.2 Statistical Methods

This section reviews the application of statistical methods in hardware design space exploration and performance prediction.

2.2.1 Linear Regression

Regression modeling has emerged as a powerful approach for predicting micro-architectural performance and power across various applications and configurations. Lee and Brooks [11] introduced regression models that significantly reduce the number of required simulations and enhance statistical inference, resulting in accurate predictions with low error rates.

To enable practical and comprehensive design space evaluation, Lee and Brooks [12] used regression-based modeling that showcased the capability to construct Pareto frontiers, analyze pipeline depth, and identify efficient heterogeneous core designs.

Dubach et al. [13] addressed the challenge of evaluating the vast micro-architectural design space by introducing architecture-centric machine learning and regression models. This model accurately predicts program performance and energy consumption across micro-architectural configurations.

Jia et al. [14] presented Stargazer, an automated GPU design space exploration framework based on step-wise regression modeling. Stargazer efficiently estimates program run time across the entire GPU design space, significantly reducing simulation time while maintaining high accuracy.

Sangaiah et al. [15] proposed Uncore RPD, a regression-based design space exploration methodology that models the impacts of the memory hierarchy and network-on-chip on-chip multiprocessor performance. By designing memory and NoC-specific regression models and considering the dynamic nature of the Uncore design space, Uncore RPD significantly reduced the number of required simulations for NoC design space exploration.

2.2.2 LASSO

The Least Absolute Shrinkage and Selection Operator (LASSO) has proven to be an effective technique in various domains for feature selection and regularization. We used LASSO in Chapter 5 to validate and compare the behavior of our proposed statistical method.

For example, Gautier et al. [16] introduced Spector, an OpenCL FPGA benchmark suite with optimization parameters, compiled designs, and performance metrics. LASSO is one of the statistical methods that Gautier et al. used to analyze their data.

Meng et al. [17] addressed the computational complexity of deep neural networks with RRAM crossbar-based in-memory computing. They presented a structured pruning approach using a multi-group LASSO algorithm, achieving significant compression and energy reduction while maintaining accuracy. This work highlights the synergy between algorithmic and hardware-level optimizations for efficient DNN acceleration.

2.2.3 Random Forest

Random Forest, a powerful ensemble learning technique, has been employed in various design space exploration (DSE) scenarios to address challenges and enhance prediction accuracy. For example, Liu and Carloni [18] introduced a learning-based approach for high-level synthesis (HLS) tools in DSE. Their model demonstrated superiority over existing models and incorporated transductive experimental design to accelerate convergence in DSE processes. In this work, Random Forest regression was used for HLS performance prediction and design space exploration.

Singh et al. [19] presented NAPEL, an estimation framework for near-memory computing architectures using ensemble learning. Leveraging microarchitectural parameters and application characteristics, NAPEL offered rapid early-stage design space exploration. In this work random forest regression is used for predicting instruction per cycle for near-memory computing applications. We also used random forest in Chapter 5 to validate and compare the behavior of our proposed statistical method.

2.3 Iterative Learning Methods

Iterative learning is a type of machine learning approach that involves iteratively building and refining a model based on the available data. Here, our focus is iterative model-based learning. In iterative model-based learning, the model is typically initialized with some prior knowledge, and then the model is updated using the collected data. For instance, in Chapter

5, an iterative model-based learning is used to build a surrogate model of the hardware. We justify our choice for iterative model-based learning by exposing two of its advantages. First, it can handle complex and dynamic environments by continuously adapting to the environment using the collected samples. Second, it can be more sample efficient and can learn from a smaller set of data samples. In this thesis, we used Bayesian optimization to create the models and active learning to incorporate the iterative nature. The next sections provide a preliminary background required for Chapter 4, and Chapter 5. Lo et al. [20] utilized BO to tune directives to achieve minimum latency. To do so, they construct Bayesian models of the design space to guide the optimization process and minimize the number of synthesis evaluations. Moreover, the method is extended to optimize multiple usages of the same directive within the design.

2.3.1 Bayesian Optimization (BO)

Bayesian optimization [21] represents an efficient technique for identifying the maximum of an objective function that is complex to evaluate. In hardware design contexts, where synthesizing digital circuits often consumes significant time, Bayesian Optimization proves invaluable. For instance, in [22], Mehrabi et al. suggested a framework for design space exploration for `C/C++` High-Level Synthesis that uses BO. Reagen et al. [23], used BO to optimize hardware accelerators for deep neural networks.

The BO approach operates under the assumption that the *posterior* probability of an objective function O, based on evidence sample data, is directly related to the product of the likelihood of samples and the *prior* probability of O. Let us define $\mathbf{x}_i$ as the i'th sample and $O(\mathbf{x}_i)$ as the evaluation of the objective function for that sample. According to Bayes theorem, for a collection of t samples such as $C_{1:t} = \{\mathbf{x}_{1:t}, O(\mathbf{x}_{1:t})\}$:

$$P(O|C_{1:t}) \propto P(C_{1:t}|O)P(O) \tag{2.1}$$

where $P(O|C_{1:t})$ denotes the posterior distribution. Since the posterior distribution accumulates beliefs about the objective function, it can be represented as a *surrogate function* which estimates the objective function O. Note that in theory, Bayesian optimization assumes that the objective function is continuous. However, it is possible to estimate a surrogate function for a discrete objective function by fitting a continuous posterior function that passes through the discrete points of the objective function O. This idea is also explored in [24].

In practice, the sampling process is considered noisy with a Gaussian Distribution $\mathcal{N}(0, \sigma^2)$ where σ denotes the noise that is incorporated with the observation. Likewise, the surrogate

function is also considered to be a Gaussian Process. It means that the surrogate function f_s returns a mean and a variance as opposed to a normal function which returns a scalar value.

$$f_s(\mathbf{x}) = \mathcal{GP}(m(\mathbf{x}), \mathbf{K} = k(\mathbf{x}, \mathbf{x}'))) \tag{2.2}$$

Eq (2.2) shows that the $\mathcal{GP}$ is constructed of a mean vector m and a covariance operator $\mathbf{K}$ comprising of jointly Gaussian distribution between the sample population $C_{1:t}$ and the next chosen sample $\mathbf{x}_{t+1}$. For this research, the squared exponential covariance kernel k is used to construct covariance operator $\mathbf{K}$:

$$k(\mathbf{x_i}, \mathbf{x_j}) = \exp(-\frac{|\mathbf{x_i} - \mathbf{x_j}|^2}{2}) \tag{2.3}$$

Bayesian optimization is efficient in terms of exploration and exploitation since it incorporates prior belief using jointly Gaussian kernels to help with the acquisition direction. Moreover, the search is directed using the *maximum probability of improvement* ($\mathcal{PI}$) function:

$$\mathcal{PI}(\mathbf{x}) = P(O(\mathbf{x^{next}}) \geq O(\mathbf{x})) = \Phi\left(\frac{m(\mathbf{x}) - O(\mathbf{x^{next}})}{\sigma(\mathbf{x})}\right) \tag{2.4}$$

where $O(\mathbf{.})$ is the objective function sample, Φ is the cumulative Gaussian distribution, and $m(\mathbf{x})$ and $\sigma(\mathbf{x})$ are the mean and variance of the samples respectively.

Algorithm 1: Bayesian Optimization Algorithm

for $t = 1, 2, 3...$ ***Max Iterations*** **do**
- **Step 1.** Find $\mathbf{x}_t$ by maximizing acquisition function $\mathcal{PI}$ in eq (2.4) over current sample collection $C_{1:t-1} = \{\mathbf{x}_{1:t-1}, O(\mathbf{x}_{1:t-1})\}$
- **Step 2.** Sample the objective function $O(\mathbf{x}_t)$
- **Step 3.** Augment the data to the sample collection $C_{1:t}$
- **Step 4.** Calculate new covariance operator using eq (2.3)
- **Step 5.** Update $\mathcal{GP}$ using eq (2.2)

Algorithm 1 shows the pseudo-code of a BO method. For simplicity, it is divided into five steps. Step 1 is the guided search using the acquisition function. Step 2 and Step 3 show the noisy sampling and augmenting new samples to the sample population. Finally, Steps 4 and 5 show the Gaussian process update procedure.

Figure 2.1 demonstrates a simple Gaussian process with three observations. The black solid line is the $\mathcal{GP}$ surrogate mean prediction of the objective function given the data, shown as

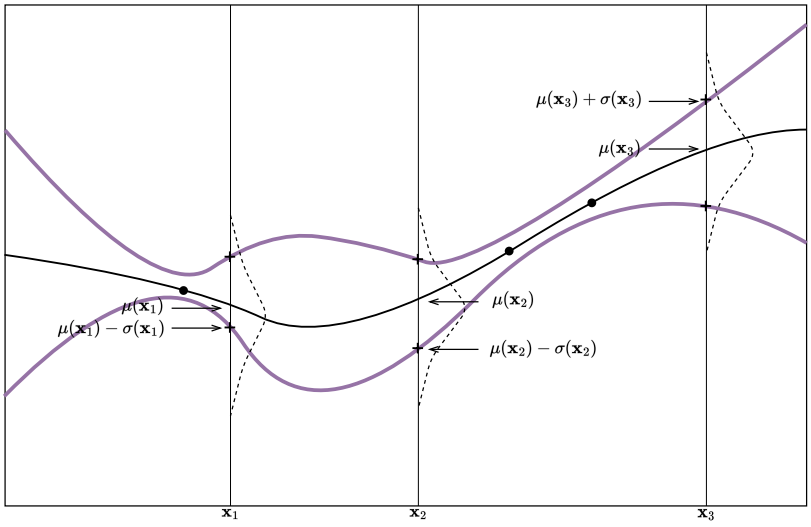


Figure 2.1: A simple 1-D Gaussian process. Adapted from [21]

m in eq (2.2). Also, the purple lines are the mean plus and minus the variance that is noted as the elements of covariance operator $\mathbf{K}$ in eq (2.2).

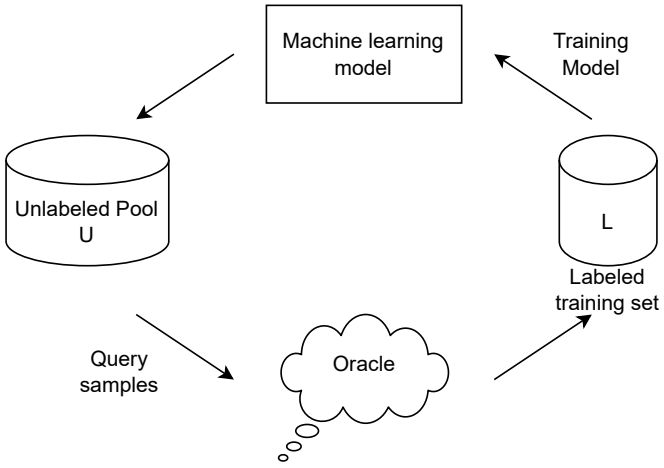


Figure 2.2: Active learning

2.3.2 Active Learning

Active learning is a class of machine learning algorithms that involves selecting a subset of informative samples to train an accurate model. The main goal of active learning is to improve the effectiveness of the learning process by selecting the samples that are most suited for it. Active learning chooses samples from a large unlabeled dataset and samples them using an *oracle*. The oracle can be a human annotator or in our case, it is a hardware synthesizer or actual performance measurement of the hardware. Figure 2.2 shows an overview of active learning where a subset of unlabeled data is labeled by oracle and sent to a machine learning model for training. One of the main advantages of active learning is that it can dramatically reduce the number of samples needed to perform the training procedure.

Active learning has been widely beneficial for various domains of machine learning applications including computer vision [25], natural language processing [26] and bioinformatics [27].

2.3.3 Reinforcement Learning

Other iterative methods such as Reinforcement Learning (RL) [28] are also effective for designing space exploration applications. An RL agent tunes hardware parameters through some iterations to maximize the reward function related to metrics of interest. For example, de Prado et al. [29] used the RL method to explore parameters that optimize the latency of deep neural networks on the ARM-Cortex-A CPUs. Likewise, Ghaffari et al. [30] suggested a time-limited reinforcement learning [31] to perform design space exploration for deeply pipelined OpenCL kernels of the convolutional neural networks. Hosny et al. [32] proposed an Advantage Actor Critic (A2C) [33] design space exploration to minimize area subject to a timing constraint in logic synthesis. Additionally, Kao et al. [34] proposed an RL-based design space exploration method and used the performance predicted by their cost model to compute the reward values. Moreover, they augmented their RL algorithm with a genetic algorithm to improve the design space exploration results.

2.4 Dominant Solution Sets and Pareto Frontiers

Multi-objective optimization in hardware design addresses the challenge of optimizing conflicting objectives like throughput, power consumption and area utilization. Instead of seeking a single optimal solution, this approach aims to identify the Pareto frontier, a set of dominant solutions representing trade-offs where enhancing one objective comes at the expense of others. These dominant solutions on the Pareto frontier offer a range of trade-offs, enabling informed design decisions that align with specific priorities and constraints, contributing to

well-balanced and efficient hardware designs.

Upon deploying an optimization method, the identification of a collection of dominant solutions, also referred to as Pareto optimal solutions, becomes an important step. To facilitate the comparative analysis of the dominant solutions achieved through diverse optimization methods, approaches such as General Distance and Average Distance to Reference Set (ADRS) [35, 36] can be used. Moreover, ADRS is commonly used in the hardware community [37, 38], serving as an essential tool for evaluating and contrasting the efficacy of different optimization techniques. This process contributes to a comprehensive assessment of the optimization strategies and their outcomes, ultimately enhancing our understanding of their relative performance.

2.5 Hardware Optimization Frameworks

Hardware optimization frameworks play an important role in the field of digital hardware design by enabling the exploration and refinement of design choices in order to achieve optimal performance, power efficiency, and other desired metrics. These frameworks facilitate the important task of tuning various design parameter sets and configurations in hardware systems, ranging from processors and accelerators to memory hierarchies and interconnects. The overall goal is to reach a balance between conflicting goals, such as maximizing throughput while minimizing energy consumption.

Numerous research articles have contributed to this domain, proposing innovative techniques,

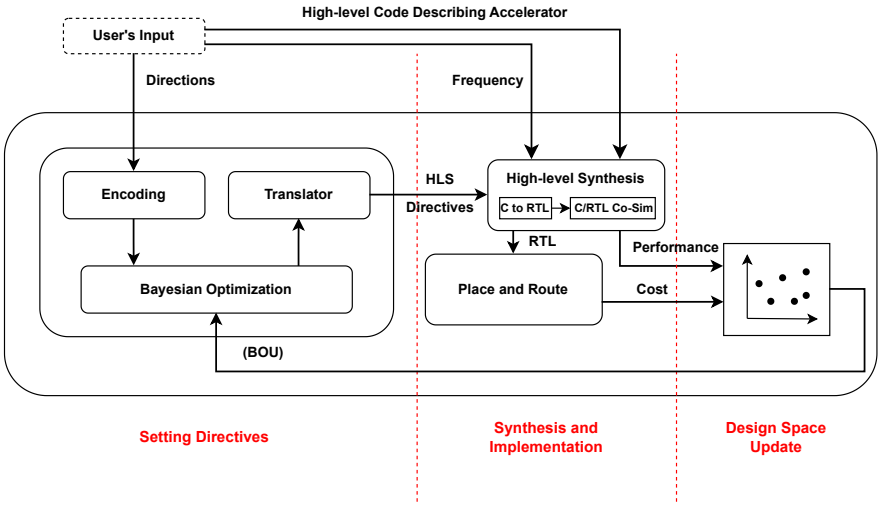


Figure 2.3: Overview of Prospector framework. Adapted from [22]

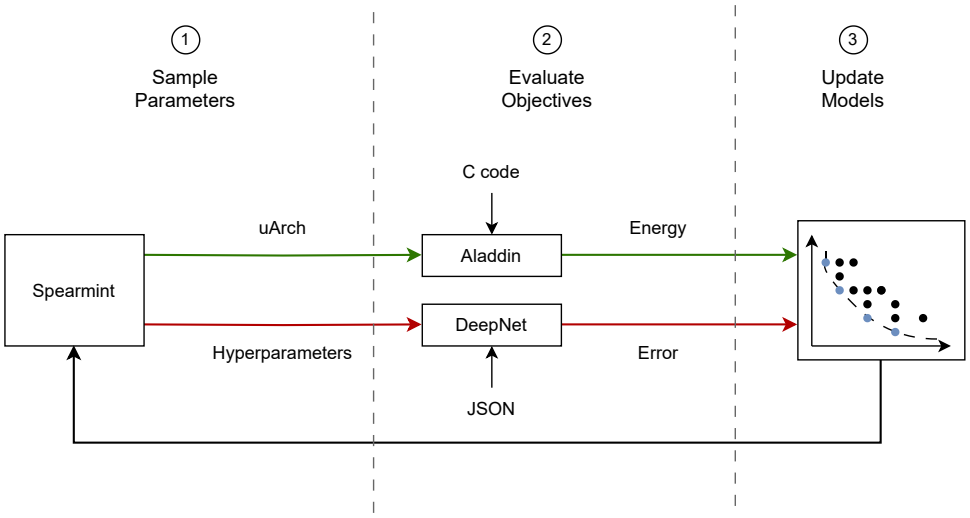


Figure 2.4: Overview of the design space exploration framework. Adapted from [23]

algorithms, and methodologies [22, 23, 37, 39, 40]. These contributions address challenges posed by the rapidly evolving landscape of technology and applications, driving the need for intelligent design exploration.

For brevity, in this section, we delve into only two representative frameworks to study their design. The related articles not only showcase state-of-the-art techniques but also shed light on the complexities and nuances associated with achieving efficient hardware designs.

For instance, Mehrabi et al. [22] suggested the Prospector framework for design space exploration using `C/C++` High-Level Synthesis. Prospector uses BO. These authors have shown that BO outperforms the traditional search method in terms of latency and resource usage in the FPGA targets. Figure 2.3 shows the overall view of the Prospector framework. This framework receives high-level source codes such as `C/C++` that describes the functionality of the hardware and produces a set of RTL implementation that reflect multiple performance trade-offs. In Prospector, the Bayesian optimization unit performs design space exploration iteratively. Here, Bayesian optimization is used to choose the synthesis directives applied to convert the high-level source codes to RTL. Using BO, Prospector is capable of identifying the Pareto optimal designs.

Similarly, in a second significant example, Reagen et al. [23] used BO to optimize hardware accelerators for deep neural networks. They reported that the BO method could help to simulate the accuracy of the deep neural network and energy efficiency of the accelerator hardware. Figure 2.4 illustrates the overview of the design space exploration framework

proposed by [23]. As shown in the figure, Spearmint [41] is used in this framework to perform Bayesian optimization, and DeepNet library [42], is also used to train and evaluate deep neural networks. Moreover, the Aladdin [43] simulator is used to estimate the energy consumption of these designs. The design flow presented in Figure 2.4 has three main steps. The first step is sampling the design parameters which is done using the Spearmint model. The second step is evaluating the objectives using DeepNet. A `json` file is used to feed design parameters to DeepNet. Finally, in the third step, the Bayesian models are updated by Spearmint using new samples gathered in the previous steps.

2.6 Comparative Study of the Literature, Limitations, and Motivations

In this section, we focus on the exploration of the existing literature that forms the backbone of the methods proposed in this thesis. Our aim is to provide a critical evaluation of the context surrounding the challenges in hardware design optimization.

2.6.1 Challenges and Opportunities in Deep Learning Algorithm Implementation

Field Programmable Gate Arrays (FPGAs) emerge as a compelling solution to address the limitations associated with GPUs while preserving algorithm accuracy. Particularly, utilizing quantized deep learning algorithms on FPGAs can alleviate power consumption and memory efficiency concerns, as the quantization process leads to smaller circuit implementations.

While various research endeavors have delved into deep learning algorithm architecture, synthesis, and optimization on FPGAs [44–48], a notable gap persists in terms of integrating these findings into a comprehensive tool. In contrast to existing tools like [49, 50], our contribution seeks to offer both integrated design space exploration and a generic CNN model analyzer. The literature reveals a specific focus on hardware design in [47, 48, 51], with limited emphasis on synthesis automation. Moreover, [45] falls short in discussing the automatic fitting of designs across diverse FPGA targets. This complexity is exacerbated by the inherent flexibility of CNN architecture, which enables designers to select convolution, pooling, and fully connected layers at their discretion. Thus, the challenge persists in developing design space exploration methods capable of optimizing FPGA resource utilization effectively. These limitations motivated us to propose our first contribution in Chapter 3.

2.6.2 Challenges and Opportunities in Optimizing OpenCL kernels for FPGA Targets

Our presented approach in Chapter 4 builds upon the utilization of Bayesian Optimization (BO) for design space exploration. However, our work distinguishes itself from existing research, such as [22,23], in two pivotal aspects that underscore its novelty and applicability.

Primarily, our approach is underpinned by High-Level Synthesis (HLS) tailored for OpenCL kernels. This application introduces a fresh set of challenges due to the intricate control of fine-grained thread-level parallelism that OpenCL kernels in FPGA devices entail. Unlike previous work, we embrace these challenges, fostering a comprehensive approach to design space exploration within the realm of OpenCL kernels and HLS.

Moreover, in contrast to prior studies, our work refrains from disregarding the potential of meta-heuristic methods. Rather than focusing solely on Bayesian Optimization, we embrace the synergy of diverse techniques. Our methodology encompasses the integration of Grey Wolf Optimization (GWO) [52], a novel meta-heuristic approach that aligns with the demands of design space exploration. This integration of GWO and BO marks a pioneering endeavor, exploiting the strengths of both methods. We introduce the details of our proposed methods in Chapter 4.

2.6.3 Addressing Limitations of Bayesian Modeling and Enhancing its Capabilities

While Bayesian models have showcased impressive capabilities in driving design space exploration and capturing parameter inter-dependencies, they encounter fundamental limitations that call for innovative solutions. The foremost challenge revolves around the absence of a feedback mechanism from real-world applications, thereby hampering the dynamic enhancement of Bayesian models' accuracy. Additionally, the application-specific nature of Bayesian models restricts their adaptability across different tasks, warranting the development of a robust *transfer learning* strategy. Particularly in the context of hardware design, such an approach becomes imperative to enable the exploration of emerging applications using insights garnered from prior experiences and tasks.

Our contribution in Chapter 5 proposes a novel methodology designed to address the identified shortcomings. Encompassing a Bayesian modeling approach, our methodology leverages Bayesian active learning [21, 53], an iterative machine learning algorithm. By actively selecting informative samples, the Bayesian active learning process incrementally refines the surrogate model of the objective function, gradually unraveling the underlying data structure.

This iterative progression culminates in a surrogate function closely mirroring the system's behavior, achieved with minimal sample expenditure.

Our proposed methodology further integrates a feedback mechanism obtained from real-world applications, thereby enhancing the accuracy of Bayesian surrogate models. By updating models using performance metrics collected from processors or computing clusters, our approach bridges the gap between theoretical modeling and real-world scenarios. Building on the notion of *transfer learning*, we extend the utility of Bayesian models by allowing the transfer of knowledge from one domain to another. This strategic knowledge transfer significantly bolsters the predictive accuracy of the statistical model in scenarios with limited prior information, a prominent challenge in hardware design for novel applications.

To achieve more efficient utilization of samples and to simulate hardware performance prediction, we introduce *Gaussian regression bootstrapping*. This technique enables the generation of new simulated samples from existing Bayesian models, facilitating diverse statistical analyses of hardware models. Our study substantiates the efficacy of Gaussian regression bootstrapping in yielding results that closely align with analyses conducted on real data. Overall, this contribution fills critical gaps in Bayesian modeling for hardware design optimization, presenting a comprehensive approach that not only enhances accuracy but also maximizes efficiency in predicting hardware performance.

CHAPTER 3 ARTICLE 1 : CNN2GATE: AN IMPLEMENTATION OF CONVOLUTIONAL NEURAL NETWORKS INFERENCE ON FPGAS WITH AUTOMATED DESIGN SPACE EXPLORATION

This chapter is a reiteration of a journal manuscript published in *Electronics* on December 21, 2020.

- Ghaffari, A. and Savaria, Y., 2020. "CNN2Gate: An Implementation of Convolutional Neural Networks Inference on FPGAs with Automated Design Space Exploration." Electronics, 9(12), doi: 10.3390/electronics9122200.

3.1 Abstract

Convolutional Neural Networks (CNNs) have a major impact on our society because of the numerous services they provide. These services include, but are not limited to image classification, video analysis, and speech recognition. Recently, the number of researches that utilize FPGAs to implement CNNs are increasing rapidly. This is due to the lower power consumption and easy reconfigurability offered by these platforms. Because of the research efforts put into topics such as architecture, synthesis and optimization, some new challenges are arising to integrate suitable hardware solutions to high-level machine learning software libraries. This paper introduces an integrated framework (CNN2Gate) that supports compilation of a CNN model for an FPGA target. CNN2Gate is capable of parsing CNN models from several popular high-level machine learning libraries such as Keras, Pytorch, Caffe2 etc. CNN2Gate extracts computation flow of layers, in addition to weights and biases and applies a "given" fixed-point quantization. Furthermore, it writes this information in the proper format for the FPGA vendor's OpenCL synthesis tools that are then used to build and run the project on FPGA. CNN2Gate performs design space exploration and fits the design on different FPGAs with limited logic resources automatically. This paper reports results of automatic synthesis and design space exploration of AlexNet and VGG-16 on various Intel FPGA platforms.

3.2 Introduction

The impact of machine learning and deep learning is rapidly growing in our society due to their diverse technological advantages. Convolutional Neural Networks (CNNs) are among

the most notable architectures that provide a very powerful tool for many applications such as video and image analysis, speech recognition and recommender systems [54]. On the other hand, CNNs require considerable computing power. In order to better satisfy some given requirements, it is possible to use high-performance processors like graphic Processing Units (GPUs) [55]. However, GPUs have some shortcomings that limit their usability and suitability in day-to-day mission-critical and real-time scenarios. The first downside of using GPUs is their high power consumption. This makes GPUs hard to use in robotics, drones, self-driving cars and Internet of Things (IoT) while these fields can highly benefit from deep learning algorithms. The second downside is the lack of external Inputs/Outputs (I/Os). GPUs are typically accessible through some PCI-express bus on their host computer. The absence of direct I/Os makes it hard to use them in mission-critical scenarios that need prompt control actions.

In particular, the floating-point number representation of CPUs and GPUs is used for most deep learning algorithms. However, it is possible to benefit from custom fixed-point numbers (quantized numbers) to reduce the power consumption, circuit footprint and to increase the number of compute engines [56]. It was proven that many convolutional neural networks can work with 8-bit quantized numbers or less [57]. Hence, GPUs waste significant computing power and electrical power consumption when performing inference in most deep learning algorithms. A better balance can be restored by designing application specific computing units using quantized arithmetic. Quantized numbers provide other benefits such as memory access performance improvements as they allocate less space in the external or internal memory of compute devices. Memory access performance and memory footprint are particularly important for hardware designers in many low-power devices. Most of the portable hardware devices such as cell-phones, drones and IoT sensors require memory optimization because of the limited resources available on these devices.

Field Programmable Gate Arrays (FPGA) can be used in these scenarios to tackle the problems caused by limitations of GPUs without compromising the accuracy of the algorithm. Using quantized deep learning algorithms can solve the power consumption and memory efficiency issues on FPGAs as the size of the implemented circuits shrinks with quantization. In [58] the authors reported that the theoretical peak performance of 6-bit integer matrix multiplication (GEMM) in the Titan X GPU is almost 180 GOP/s/Watt, while it can be as high as 380 GOP/s/Watt for Stratix 10, and 200 GOP/s/Watt for Arria 10 FPGAs. This means high end FPGAs are able to provide comparable or even better performance per Watt than the modern GPUs. FPGAs are scalable and configurable. Thus, a deep convolutional neural network can be configured and scaled to be used in a much smaller FPGA in order to reduce power consumption for mobile devices. The present paper investigates various de-

sign space exploration methods that can fit a desired CNN to a selected FPGA with limited resources. Having massive connectivity through I/Os is natural in FPGAs. Furthermore, FPGAs can be flexibly optimized and tailored to perform various applications in industrial and real-time mobile workloads [59]. FPGA vendors such as Intel offer a range of commercialized System-on-Chip (SoC) devices that integrate both processor and FPGA fabric into a single chip (see, for example, [60]). These SoCs are widely used on mobile devices and they make using of application-specific processors obsolete in many cases .

Several research contributions address architecture, synthesis and optimization of deep learning algorithms on FPGAs [44–48]. Nonetheless, little work was done about integrating the results into a single tool. An in-depth literature review in Section 3.3 exposes that, tools such as [49, 50] provide neither integrated design space exploration nor generic CNN model analyzer. In [47, 48, 51] the authors specifically discuss the hardware design and not the automation of the synthesis. In [45], fitting the design automatically in different FPGA targets is not discussed. This presents a major challenge as there are many degrees of freedom in designing a CNN. There is no fixed architecture for CNN as a designer can choose as many convolution, pooling and fully connected layers as needed to reach the desired accuracy. In addition, there is a need for design space exploration methods that optimize resources utilization on FPGA.

A number of development environments exist for the description and architectural synthesis of digital systems [61, 62]. However, they are designed for general purpose applications. This means that there is a great opportunity to make a library which uses these hardware design environments to implement deep learning algorithms. Alternately, there are also many other development frameworks/libraries for deep learning in Python language. Some notable libraries for deep learning are Pytorch, Keras, Tensorflow and Caffe. Another challenge would be designing a synthesis tool for FPGA that can accept models produced by all the aforementioned Python libraries and many more available for machine learning developers.

The present paper proposes methods to tackle research challenges related to the integration of high-level synthesis tools for convolutional neural networks on FPGAs. The contributions of the present paper are:

1. **Generalized Model Analysis**

   Most of the previous implementations of the CNNs on FPGA fall short in supporting as many machine learning libraries as possible. CNN2Gate benefits from a generalized model transfer format called ONNX (Open Neural Network eXchange format) [63]. ONNX helps hardware synthesis tools to be decoupled from the framework in which a

specific CNN was designed. Using ONNX in CNN2Gate provides us the ability to focus on hardware synthesis without being concerned by details of specific machine learning tools. CNN2Gate integrates an ONNX parser that extracts the computation data-flow as well as weights and biases from the ONNX representation of a CNN model. It then writes these data in a format that is more usable with hardware synthesis workflows.

2. **Automated High-level Synthesis Tool**

   Several hardware implementation aspects of machine learning algorithms are not part of skill set of many machine learning engineers and computer scientists. High-level synthesis is one of the solutions that aims at improving hardware design productivity. CNN2Gate offers a high-level synthesis workflow that can be used to implement CNN models on FPGA. CNN2Gate is a Python library that parses, gets the design synthesized using vendor's OpenCL tools and runs a CNN model automatically. Its embedded procedures aim to achieve the best throughput performance. Furthermore, CNN2Gate eliminates the need for FPGA experts to manually implement the CNN model targeting FPGA hardware during early stages of the design. Having the form of a Python library, CNN2Gate can be directly exploited by machine learning developers to implement a CNN model on FPGAs. CNN2Gate is built around an available open-source tool, notably *PipeCNN* [44] that exploits the capabilities of existing design tools to use OpenCL kernels from which high-level synthesis can be performed.

3. **Hardware-aware Design Space Exploration**

   An important aspect of design space exploration is to try choosing design parameters that achieve some desired performance prior to generation of physical design. CNN2Gate provides a design space exploration tool that is able to adjust the level of parallelism of the algorithm to fit the design on various FPGA platforms. The design space exploration methods proposed here use estimations of hardware resource requirements (e.g. DSPs, lookup tables, registers and on-chip memory) to fit the design. These estimates are obtained by invoking the first stage of the synthesis tool provided by the FPGA vendor and receives back the estimated hardware resource utilization. In the next step, the tool tunes the design parameters according to the resource utilization feedback and iterates again to get the new hardware resource utilization. We have implemented three algorithms to do design space exploration. The first algorithm is based on brute-force to check all possible parameter values. The second algorithm is based on a reinforcement learning agent that explores the design space with a set of defined policies and actions. Finally, the third algorithm uses the hill climbing method to tackle the problem of design space exploration in large convex design-spaces. Ad-

vantages and disadvantages of these three exploration algorithms will be discussed in the corresponding sections.

Note that CNN2Gate emphasizes **software/hardware** co-design methods. The goal of this paper is not to compare the performance of FPGAs and GPUs, but it explores the possibility of designing end-to-end generic frameworks that can leverage high-level model descriptions to implement FPGA-based CNN accelerators without human intervention. As shown in Figure 3.1, CNN2Gate is a Python library that can be used to perform inference of CNNs on FPGAs that is capable of:

- Parsing CNN models.
- Extracting the computation flow of each layer.
- Extracting weights and biases of each kernel.
- Applying post-training quantization values to the extracted weights and biases.
- Writing this information in the proper format for hardware synthesis tools.
- Performing design space exploration for various FPGA devices using a hardware-aware smart algorithm.
- Synthesizing and running the project on FPGA using the FPGA vendor's synthesis tool.

The rest of the paper is organized as follows. Section 3.3 reviews the related works. Section 3.4 reviews the most relevant background knowledge on convolutional neural networks. Section 3.5 elaborates on how CNN2Gate extracts the computation flow, configures the kernels and does design space exploration. Then, Section 3.6 reports some results and compares them to other existing implementations.

3.3 Related Works

A great deal of research was conducted on implementing deep neural networks on FPGAs. Among those researches *hls4ml* [50], *fpgaConvNet* [45] and *Caffeine* [51] are the most similar to the present paper. hls4ml is a companion compiler package for machine learning inference on FPGA. It translates open-source machine learning models into HLS descriptions. hls4ml was specifically developed for an application in particle physics, with the purpose of reducing development time on FPGA. However, as stated in the status page of the project [64], the

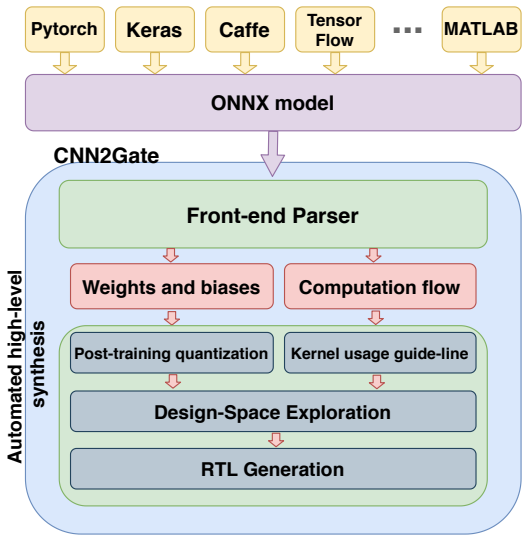


Figure 3.1: CNN2Gate overall architecture comprising a front-end parser (ONNX parser), a design space exploration module, and leverages automated high-level synthesis

package only supports Keras and Tensorflow for CNNs and the support for Pytorch is in development. In addition, to the best of our knowledge, hls4ml does not offer design space exploration. FpgaConvNet is also an end-to-end framework for the optimized mapping of CNNs on FPGAs. FpgaConvNet uses a symmetric multi-objective algorithm to optimize the generated design for either throughput, latency, or multi-objective criteria (e.g. throughput and latency). The front-end parser of fpgaConvNet can analyze models expressed in Torch and Caffe machine-learning libraries. Caffeine is also a software-hardware co-design library that directly synthesize Caffe models comprising convolutional layers and fully connected layers for FPGAs. The main differences between CNN2Gate and other cited works are in three key features. First, as explained in Section 3.5.1, CNN2Gate leverages a model transfer layer (ONNX), which automatically brings support for the most machine-learning Python libraries without bounding the user to a specific machine-learning library. Second, CNN2Gate is based on OpenCL unlike hls4ml and fpgasConvNet that are based on C++. Third, CNN2Gate proposes an FPGA fitter algorithm based on reinforcement learning.

Using existing high-level synthesis technologies, it is possible to synthesize OpenCL Single Instruction Multiple Thread (SIMT) algorithms to RTL. It is worth mentioning some notable research efforts in that direction. In [65], the authors provided a deep learning accelerator targeting Intel's FPGA devices based on OpenCL. This architecture was capable of maximizing data-reuse and minimizing memory accesses. The authors of [66] presented a systematic design space exploration methodology to maximize the throughput of an OpenCL-based FPGA

accelerator for a given CNN model. They used synthesis results to empirically model the FPGA resource utilization. Similarly, in [67], the authors analyzed the throughput and memory bandwidth quantitatively to tackle the problem of design space exploration of a CNN design targeting FPGAs. They also applied various optimization methods such as loop-tiling to reach the best performance. A CNN RTL compiler is proposed in [68]. This compiler automatically generates sets of scalable computing primitives to form an end-to-end CNN accelerator.

Another remarkable OpenCL-based implementation of CNNs is *PipeCNN* [44]. PipeCNN is mainly based on the capability of currently available design tools to use OpenCL kernels in high-level synthesis. PipeCNN consists of a set of configurable OpenCL kernels to accelerate CNN and optimize memory bandwidth utilization. Data reuse and task mapping techniques have also been used in that design. Recently, in [69], the authors added a sparse convolution scheme to PipeCNN to further improve its throughput. Our work (CNN2Gate) follows the spirit of PipeCNN. CNN2Gate is built on top of a modified version of PipeCNN. CNN2Gate is capable of exploiting a library of primitive kernels needed to perform inference of a CNN. In addition, CNN2Gate also includes means to perform automated design space exploration and can automatically translate CNN models provided by a wide variety of machine learning libraries. It should be noted that our research goal is introducing a methodology to design an end-to-end framework to implement CNN models on FPGA targets. This means, without loss of generality, that CNN2Gate can be modified to support other hardware implementations or technologies ranging from RTL to high-level synthesis with two conditions. First, CNN layers can be expressed as pre-defined templates. Second, the amount of parallelism in the hardware templates can be controlled.

There are several reasons that we have chosen PipeCNN as the foundation of CNN2Gate. First, using OpenCL model, it is possible to fine-tune the amount of parallelism in the algorithm as explained in the Section 3.5.3. Second, it supports the possibility of having deep pipelined design as explained in Section 3.4.2. Third, the library of primitive kernels can be easily adapted and re-configured based on the information extracted from a CNN model (Section 3.5.1).

Lately, reinforcement learning has been used to perform automated quantization for neural networks. *HAQ* [70] and *ReLeQ* [71] are examples of research efforts exploiting this technique. ReLeQ proposes a systematic approach to solve the problem of deep quantization automatically. This provides a general solution for quantization of a large variety of neural networks. Likewise, HAQ, suggests a hardware-aware automated quantization method for deep neural networks using actor-critic reinforcement learning method [72]. Inspired by these

two papers, we used a reinforcement learning algorithm to control the level of parallelism in CNN2Gate OpenCL kernels.

Finally, it is worth mentioning the challenges of implementing digital vision systems in the hardware. In [73], the authors proposed a hardware implementation of haze removal method exploiting adaptive filtering. Also in [74] the authors provided an FPGA implementation of a novel method to recover clear images from degraded ones on FPGA.

3.4 Background

This section provides the background knowledge required to understand CNNs and OpenCL-based high-level synthesis workflows.

3.4.1 Convolutional Neural Networks (CNNs)

CNNs are categorized as feedforward networks. Figure 3.2 shows the most common architecture of a CNN. A CNN consists of one or several convolutional, pooling and fully connected layers. These layers can be connected together to shape a deep convolutional neural network model. The data enters from the input, is processed by various hidden layers and the classification results are presented as outputs.

Convolutional Layers

Convolutional layers are used to extract features from the input data. A convolutional layer convolves the input with a convolutional kernel (filter) and passes the results to the next layer through a non-linear activation function. More specifically, Each neuron in convolutional layers has a receptive field and is connected to other neurons in the adjacent layers through some learned weights and biases. The following equation shows that the feature F_k can be computed as:

$$F_k = f\Big(W_k * I + b_k\Big) \tag{3.1}$$

in which I is the input from the preceding layer or input image and W_k is the convolution kernel for feature k and b_k is the bias vector. The non-linear activation function is denoted $f(.)$ in (3.1).

Pooling Layers

Pooling layers are used in CNNs to down-sample and reduce the spatial resolution of the extracted features. Reduction of spatial resolution can help a CNN model to overcome input

distortion and angular transformations. Pooling layers are usually made of max-pooling kernels. A max-pooling kernel selects the maximum value of the region of interest. The max-pooling feature can be defined as:

$$\mathrm{MPF}_k = \max_{i,j \in \mathfrak{R}_{\mathrm{m,n}}} x_{k,(i,j)} \tag{3.2}$$

where MPF_k denotes k'th max-pooling feature and $\mathfrak{R}_{\mathrm{m,n}}$ shows the region of interest around point (m, n).

Fully Connected Layers

After extracting features of the input data by convolutional and pooling layers, the data is sent to a fully connected neural network that implements the decision making process. The fully connected layer interprets the extracted features and turn them into information represented with quantities. A softmax function is often used at the output of a fully connected layer to classify the output.

For more information on CNNs, readers are encouraged to read papers such as [54, 75].

3.4.2 OpenCL-based High-level Synthesis of CNNs

OpenCL High-level Synthesis on FPGAs

OpenCL can be used to write programs that can be executed across heterogeneous platforms. OpenCL offers the ability to describe a parallel algorithm to be implemented on FPGA, for example.

However, this parallel description is at a level of abstraction much higher than hardware description languages such as VHDL. An OpenCL application consists of two parts. The OpenCL "host" program that is written purely in C or C++ and that can be executed on any type of processor. On the other hand, "kernels" are accelerated functions that are implemented on some co-processor "device", such as an FPGA, using fine-grained parallelism. The host can offload computation workload to kernels using sets of command queues. This concept is demonstrated in Figure 3.3.

OpenCL Pipes

As depicted in Figure 3.4a, in the OpenCL model, every kernel has access to the global memory of the device. In the case of massively parallel algorithms, memory transactions

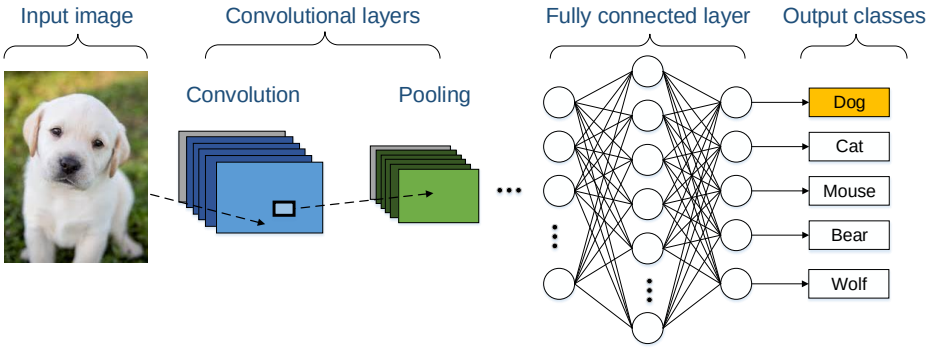


Figure 3.2: Demonstration of a Convolutional Neural Network (CNN) comprising an input image, convolution and pooling layers, fully connected layer and output classification

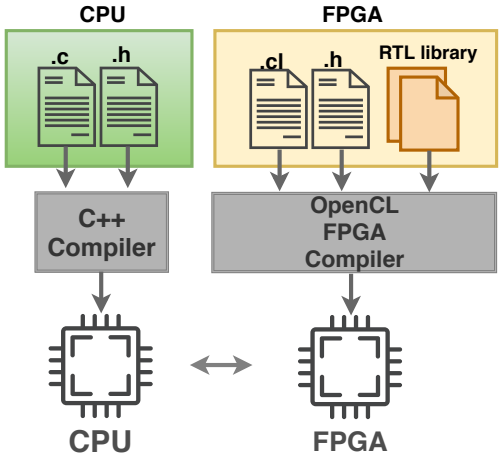


Figure 3.3: OpenCL high-level synthesis for FPGA

can be a bottleneck. In OpenCL 2.0 [49], "Pipes" were introduced to enable kernel-to-kernel communications. These tiny communication channels between kernels help to reduce the number of times kernels refer to memory and increases the memory access efficiency. In the case of FPGAs, these channels are implemented using FIFOs. Figure 3.4b shows that it is possible to stack OpenCL kernels on top of each other and pass data from one kernel to another. This feature makes FPGAs very well suited to implement stacked layers of CNNs as deeply pipelined kernels.

OpenCL High-level Synthesis of CNNs for FPGAs

Leveraging the view from Figure 3.4b, a CNN model can be synthesized on FPGA using deeply pipelined kernels as shown in Figure 3.4c. For this architecture, the following hardware

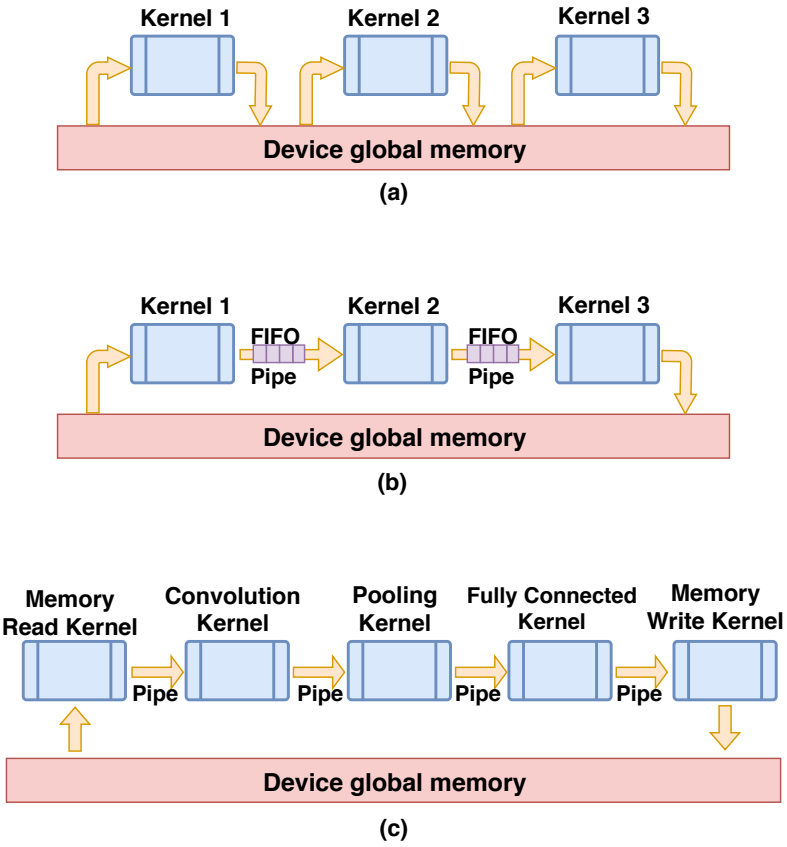


Figure 3.4: (a) Memory access pattern in OpenCL standard 1.x. (b) OpenCL pipes; in FPGAs, pipes are implemented as FIFOs. (c) Deeply pipelined CNN network architecture

accelerator kernels are needed: 1) Memory read 2) Memory write 3) Convolution 4) Pooling and 5) Fully connected. In many cases, convolution kernels and fully connected kernels can be fused together as a single 3-D matrix-matrix multiplication unit. Memory read/write kernels provide and store data for other kernels. This architecture has two main advantages: 1) Depending on the number of computing units in the convolution and pooling layer and the size of data fetched by memory read/write kernels, this architecture can be scalable (details will be discussed in the design space exploration section) and 2) pipelined kernels can process data without storing the data moved between layers; this can significantly improve memory access efficiency.

3.5 Proposed Architecture

3.5.1 Generalized Model Analysis Using ONNX

ONNX is a library that makes deep learning models interoperable. ONNX represents the computation flow of a deep neural network as an extensible computation graph model using its own built-in operators and data-types. This inter-operable ecosystem eliminates limitations that may stem from using any specific deep learning development framework. This makes it easier for hardware developers to exploit the most suited tool without being bound to the library with which the model was developed. ONNX provides the definition of a deep neural model using extensible acyclic graphs. Nodes represent an operator and have sets of inputs and outputs. ONNX can support a vast variety of tools such as Pytorch, TensorFlow, Caffe, Keras and more [63]. CNN2Gate offers a front-end ONNX parser for CNNs. As shown in Figure 3.1, using ONNX as a transport layer decouples the high-level synthesis tool from the machine learning tool. CNN2Gate parses the computation dataflow –or the arrangement of layers– besides weights and biases for each layer. The CNN2Gate parser traverses the ONNX graph nodes and extracts the synthesis information of each node based on the following operator types:

- **Convolution**: For the convolution operator, CNN2Gate parses dilations, pads, kernel shape, and stride. The reader can refer to [76] for more information about these variables and how they affect the computation. It also extracts the learned weights and biases for convolutional kernels. CNN2Gate also computes the output tensor size of the layer using eq (3.3). Let us assume the input of a two dimensional convolutional kernel is of size $(c_{\text{in}}, h_{\text{in}}, w_{\text{in}})$ where c denotes the number of features, h denotes the height and w denotes the width. The output tensor size $(c_{\text{out}}, h_{\text{out}}, w_{\text{out}})$ can be written as:

$$\begin{aligned} h_{\text{out}} &= \left\lfloor \frac{h_{in} + 2p[0] - d[0](ks[0] - 1) - 1}{st[0]} + 1 \right\rfloor \\ w_{\text{out}} &= \left\lfloor \frac{w_{in} + 2p[1] - d[1](ks[1] - 1) - 1}{st[1]} + 1 \right\rfloor \end{aligned} \tag{3.3}$$

$$c_{\text{out}} = c_{\text{in}} \tag{3.4}$$

  where ks is the kernel size, st is the stride, while p and d are the padding and dilation parameters respectively.

- **Max-pooling**: Similar to the convolution, CNN2Gate parses dilations, pads, kernel size, and strides. However, as max-pooling is a down-sampling kernel, it does not have

weights and biases. The output tensor size of a max-pooling node with input size of $(c_{\text{in}}, h_{\text{in}}, w_{\text{in}})$ is identical to equations (3.3) and (3.4).

- **ReLu**: CNN2Gate detects the presence of activation function such as "Relu" after a convolutional or max-pooling layer.
- **General Matrix Multiplication (GEMM)**: A fully connected layer appears as a GEMM operator in ONNX dataflow graph. In CNN2Gate, there is no specific kernel for the fully connected layer.
- **Softmax**: CNN2Gate detects the presence of the softmax operator after a fully connected layer.

The front-end parser saves the information that specifies each layer's data in a linked structure to preserve the order. Later, this data structure is used by a high-level hardware synthesis tool. The preserved order serves as a guideline for the synthesizer to configure hardware pipelines.

3.5.2 Automated High-level Synthesis Tool

Inspired from the Gajski-Kuhn chart [77], Figure 3.5 sketches the basic idea behind the CNN2Gate automated synthesis workflow. According to this diagram, the design of CNN2Gate is projected in three domains. The Algorithmic domain axis depicts the definition of concurrent algorithms, the Structural domain axis shows the building blocks to realize the Algorithmic domain. Finally, the Physical domain corresponds to the actual implementations in RTL. The lines connecting the dots show the relations between these domains. In the "Algorithmic

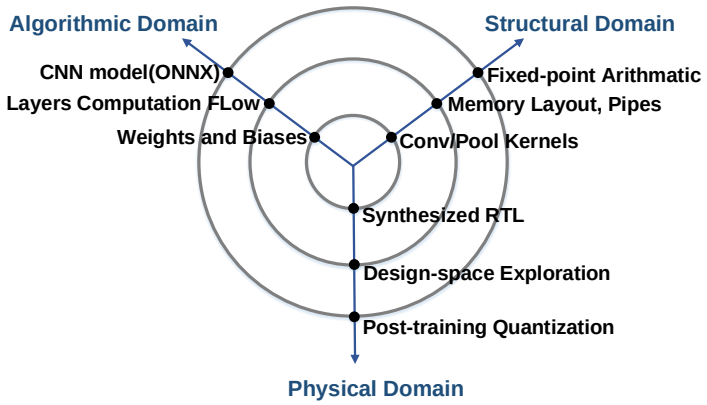


Figure 3.5: Gajski-Kuhn chart illustrating different aspects of CNN2Gate

Domain", CNN2Gate parses the information from an ONNX model as explained in Section 3.5.1. In the "Structural Domain", CNN2Gate uses 8-bit fixed point arithmetic units to perform computations. In addition, it configures memories, pipes and kernels corresponding to the information that is received from the ONNX model. In the "Physical Domain", if weights and biases are floating point numbers, CNN2Gate can quantize these values based on the information that the user provides from post-training quantization [56]. To clarify further, CNN2Gate does not perform quantization itself, however, it can apply a given value that the user provides for a layer. This value can be expressed as an (N, m) pair where fixed-point weights/biases values are represented as $N \times 2^{-m}$. Moreover, CNN2Gate performs design space exploration and generates Register Transfer Level (RTL) models targeting FPGAs.

Note that CNN2Gate can automatically configure several memory buffers depending on the layer operation type. For instance, if the next layer in the neural network is fully connected layer, it writes the data to the memory buffer associated with the fully connected layers and similarly, if the layer is convolutional, it writes the data to the convolution buffer.

CNN2Gate is also capable of building and running the CNN model in both emulation and full flow mode. For the emulation mode, CNN2Gate compiles the project for a CPU. In some cases, the user needs to verify if the CNN model performs correctly in terms of computational accuracy before committing to long hours of synthesis. The compilation for the emulation mode is significantly faster – in the order of seconds – as compared to synthesizing the full flow for the FPGA, which takes several hours. This feature makes the workflow more versatile for the designers who want to iterate between a FPGA design and a CNN model to reach the best quantization parameters and accuracy. To synthesize the full flow on FPGA, CNN2Gate accepts the name of the FPGA board to perform design space exploration (Section 3.5.3) and generates the RTL accordingly. To validate CNN2Gate, we tested this process by targeting three different Intel™ FPGA boards [78–80] and report the results later in this paper. In addition, we used Intel OpenCL SDK 16.1 to synthesize our designs.

Taking advantage of OpenCL flexibility, it is possible to design an architecture with several degrees of freedom. The first degree of freedom is the size of the pipes. The better throughput of the pipes means less congestion point for data to be moved from a kernel to another. The second degree of freedom is the bandwidth of data that memory write/read kernels provide. The third degree of freedom is the number of parallel convolutional (CONV) and RELU units that are used to implement convolution kernels. Figure 3.6 shows this concept in a simple example. These degrees of freedom for deeply pipelined kernels are leveraged from what proposed by [44]. The memory read kernel fetches N_l vectors of size N_i for features and weights. Tuning N_i and N_l can provide a better throughput for data write/read kernels.

Note that the memory access schedule of where and when to read the features and weights are derived by the host program. The memory access schedule is configured by the front-end parser based on the CNN model . The number of computation lanes (N_l) shows the level of parallelism of the algorithm. The number of CONVs in a convolution kernel, the size of data pipes and the number of max-pool operators in the max-pool kernel are tuned according to N_l. Changing N_l and N_i can result in different utilization ratios of FPGA resources. For instance, in the case of increasing N_l, 1) more on-chip memory in read/write kernels, 2) more register for pipe FIFOs, 3) more DSP slices for CONVs and 4) more LUT for max-pooling kernels are needed to accommodate the design on FPGA.

Architectural Limitations

The architecture of Figure 3.6 is used to perform the calculation of all layers. In order to obtain practical implementations and to allow targeting FPGAs of various size, it is necessary to fold (or time multiplex) the layers onto fewer hardware lanes and vectors [45,46]. Therefore, arbitrary choices for N_l and N_i are not always possible. N_i should be a divisor of the features' width for all layers to avoid padding. Likewise, N_l should be a divisor of the number of features for all layers to avoid idle lanes in some layers.

3.5.3 Hardware-aware Design Space Exploration

CNN2Gate analyzes the computation flow layers that are extracted from the model by the ONNX front-end parser and determines all options possible for N_l and N_i. CNN2Gate is also capable of interacting with the Intel OpenCL compiler to estimate resource usage for a specific choice of N_l and N_i. There is thus a need to identify the best values for N_l and N_i. The process of finding the best values of N_l and N_i is what we call design space exploration. Many methods can be considered to do it and three such methods are investigated in this section.

Brute Force Design Space Exploration (BF-DSE)

This method exhaustively searches for all possible pairs of N_l and N_i and finds the feasible option that maximizes FPGA resource utilization. In our case, the solution maximizing resource utilization corresponds to the one providing the best throughput. This method is simple to execute and it always find the best solutions. However, for larger FPGAs with large number of possible candidates for N_l and N_i, an exhaustive brute force search could take excessive time. In the next section, a more efficient search strategy is described.

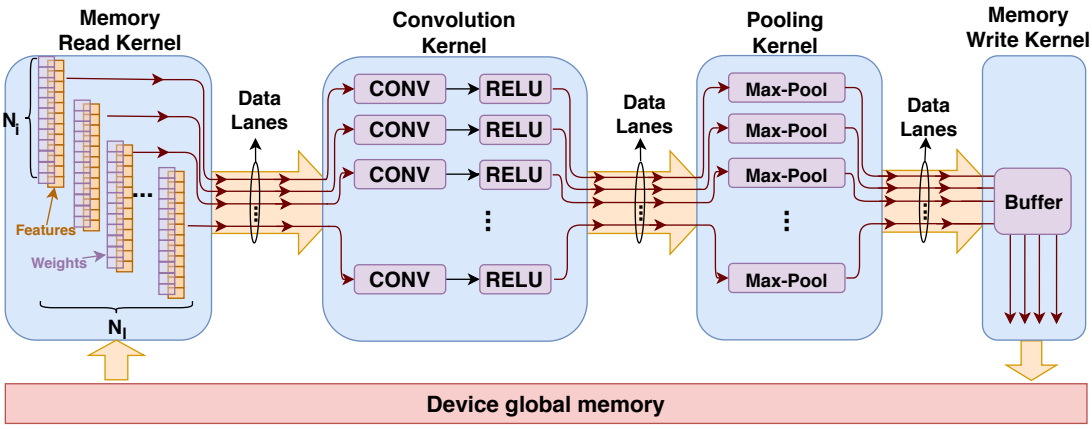


Figure 3.6: Detailed demonstration of vectorized input data and weights and the concept of computation lanes in pipelined kernels

Reinforcement Learning Design Space Exploration (RL-DSE)

RL-DSE trains a RL agent to find the best options for N_l and N_i. Reinforcement learning for design space exploration is of interest for two reasons. First, it can be faster than brute force and second, it can be merged with other RL-agents such as HAQ [70] or ReleQ [71] to determine the level of parallelism and the quantization of each layer. RL-DSE explores hardware options for N_l and N_i and finds the best fit for the specified hardware. The RL-DSE agent receives a reward signal corresponding to the feedback provided by the Intel OpenCL compiler for FPGA resource utilization. Hence, the reward function for RL-DSE is designed to maximize FPGA resource utilization.

When CNN2Gate triggers Intel OpenCL compiler to evaluate a hardware option, it receives the corresponding hardware resource utilization. This feedback comprises 1) the percentage of lookup table utilization, 2) the percentage of DSP utilization, 3) the percentage of on-chip memory block utilization and 4) the percentage of register utilization. We denote these percentages P_{lut}, P_{dsp}, P_{mem} and P_{reg} respectively.

Given the percentages of resource utilization, the agent takes a series of actions. The agent starts from the minimum values of N_l and N_i. RL-DSE can flexibly choose to 1) increase N_l, 2) increase N_i, or 3) increase both N_i and N_l. If one of the variables reaches the maximum possible value based on the CNN topology, the variable is reset to its initial value.

Let us assume that the average usage factor is defined as:

$$F_{\text{avg}} = \frac{P_{\text{lut}} + P_{\text{dsp}} + P_{\text{mem}} + P_{\text{reg}}}{4} \tag{3.5}$$

Further assume that the user defines a vector of thresholds T_{th} for the maximum usage that is tolerated for each quota. The reward function can be described as follows.

Algorithm 2: Reward shaping

Input : T_{th} , F_{avg}, P_{lut}, P_{dsp}, P_{mem}, P_{reg}, N_i and N_l

Output: Reward , H_{best}

Variables: $F_{\text{max}}, T_{\text{th}} = (T_{\text{lut}}, T_{\text{dsp}}, T_{\text{mem}}, T_{\text{reg}})$

if $(P_{\text{lut}}, P_{\text{dsp}}, P_{\text{mem}}, P_{\text{reg}}) < (T_{\text{lut}}, T_{\text{dsp}}, T_{\text{mem}}, T_{\text{reg}})$ **then**

  if $F_{\text{avg}} > F_{\text{max}}$ **then**

    $F_{\text{max}} = F_{\text{avg}}$

    Reward $= \beta \times F_{\text{avg}}$

    $H_{best} = (N_i, N_l)$

  else

    Reward $= 0$

else

  Reward $= -1$

In Algorithm 2, H_{best} is the best hardware options and F_{max} is the maximum usage observed by the agent during the exploration of the search environment. In the reward shaping function, if at least one of the hardware utilization quotas (P_{lut}, P_{dsp}, P_{mem}, P_{reg}) exceeds the thresholds specified in the vector of threshold limits ($T_{\text{th}} = (T_{\text{lut}}, T_{\text{dsp}}, T_{\text{mem}}, T_{\text{reg}})$), the agent receives a negative reward for this choice and the chance of choosing this option for a later iteration decreases. The reward function keeps track of the maximum usage score F_{max}, and constantly update F_{max} and H_{best} with the best hardware option observed by the agent in the environment. Since during exploration the best value of the reward function is unknown, besides exhaustion of the search space, there is no exact stop condition for agent exploration. In this case, we used a variation of time-limited reinforcement learning [31] with which, the number of iterations in each episode is limited. Finally, in our RL-DSE, a scaling factor $\beta = 0.01$ is applied to F_{avg} to form the final reward function to convert it from percentage scale to a number between 0 and 1.

Note that a discount factor $\gamma = 0.1$ is used in our RL agent design. The discount factor specifies that the agent does not have unlimited time to find the best option. Thus, the agent receives a down-scaled reward as it spends more time in the environment. The discount factor urges the agent to optimize the total discounted reward [81]:

$$r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + ... = \sum_{i=1}^{n} \gamma^i r_{t+i} \tag{3.6}$$

where r_t is the reward calculated in time t according to Algorithm 2.

Hill Climbing Design Space Exploration (HC-DSE)

While exhaustive enumeration of all possible hardware options (BF-DSE) is very efficient in small design-spaces, the execution time increases proportionally to the number of possibilities in the design-space. There are other smart optimization algorithms that find the best solution significantly faster than brute-force. Hill climbing is an iterative numerical algorithm that starts from an arbitrary location in the design space and attempts to find a better solution by moving incrementally to optimize the target function. As shown in Figure 3.7, optimizer O examines all possible successors (in the 2D case A, B and C) and calculates relative changes to the target function corresponding to those successors (Δ_A, Δ_B and Δ_C). Afterward, the optimizer moves toward the best choice. This process continues until there is no further improvement in the target function, or, in our case, it continues until the design does not fit on the target FPGA. This simple algorithm is very efficient in finding local optimum. Algorithm. 3 shows the pseudo code for hill climbing in our context.

Note that the hill climbing algorithm is guaranteed to find the best solution in a **convex** design-space.

Table 3.1 Execution times for Alexnet and VGG (batch size = 1)

Platform	**Resource Type**	**Execution Time**		f_{max}
		AlexNet	**VGG-16**	
Core-i7 (Emulation)	CPU	13 s	148 s	N/A
Cyclone V 5CSEMA5	System-on-Chip	153 ms	4.26 s	131 MHz
Arria 10 GX 1150	CPU+FPGA with PCIe link	18 ms	205 ms	199 MHz

3.6 Results

Table 3.1 shows the execution times of AlexNet [82] and VGG-16 [83] for three platforms using CNN2Gate. As mentioned before, the user can verify the CNN model on CPU using the CNN2Gate emulation mode in order to confirm the resulting numerical accuracy. Even if the execution time is rather large, this is a very useful feature to let the developer verify the validity of the CNN design on the target hardware before going forward for synthesis, which is a very time-consuming process. Note that the emulation's execution time cannot be a reference for the throughput performance of a core-i7 processor. The emulation mode only serves the purpose of verifying the OpenCL kernels operations. In [84], the authors

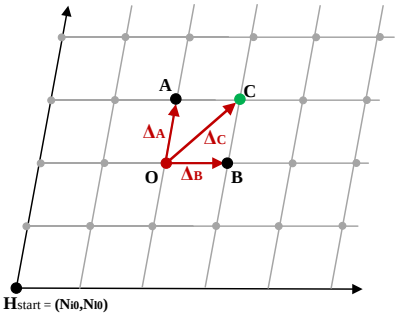


Figure 3.7: The hill-climber in a 2D design-space. Optimizer O moves toward the direction that optimizes the cost function Δ

Algorithm 3: Hill climbing

Input : $H_{\text{start}} = (N_{i_0}, N_{l_0})$
Output: H_{best}
$H_{\text{current}} := H_{\text{start}}$
compute: $F_{\text{avg}}(\text{current})$
while *true* **do**
  $H_{\text{next}} :=$ `null`
  $F_{\text{avg}}(\text{next}) :=$ `-inf`
  for i : ***Neighbours***(H_{current}) **do**
    if $(P_{\text{lut}}, P_{\text{dsp}}, P_{\text{mem}}, P_{\text{reg}})_i <$ $(T_{\text{lut}}, T_{\text{dsp}}, T_{\text{mem}}, T_{\text{reg}})$ **then**
      if $F_{\text{avg}}(i) > F_{\text{avg}}(\text{next})$ **then**
        $F_{\text{avg}}(\text{next}) = F_{\text{avg}}(i)$
        $H_{\text{next}} = H_i$
  if $F_{\text{avg}}(\text{next}) \leqslant F_{\text{avg}}(\text{current})$ **then**
    $H_{\text{best}} = H_{\text{current}}$
    return : H_{best}

described the execution of AlexNet on desktop CPUs and reported an execution time as low as 2.15 seconds. The reported results also show the scalability of this design Indeed, results are reported for both the low cost Cyclone V SoC and for the much more expensive Arria 10 FPGA that has much more resources. The exploited level of parallelism was automatically extended by the design space exploration algorithm to obtain better results for execution times commensurate with the capacity of the hardware platform.

To maintain the scalability of the algorithm, it is not always possible to use arbitrary choices for (N_i, N_l) parameters. These parameters must be chosen in a way that kernels can be used

Table 3.2 CNN2Gate Synthesis and Design Space Exploration Details (AlexNet)

Platform	HC-DSE Time	RL-DSE Time	BF-DSE Time	Synthesis Time	Resources Available	Resources Consumed	Hardware Options (N_i, N_l)
Cyclone V SoC 5CSEMA4	20 sec	2.5 min	3.5 min	N/A	**ALM:** 15 k **DSP:** 83 **RAM blocks:** 321	Does not fit	N/A
Cyclone V SoC 5CSEMA5	1 min	2.5 min	3.5 min	46 min	**ALM:** 32 k **DSP:** 87 **RAM blocks:** 397 **Mem. bits:** 4 M	**ALM:** 26 k **DSP:** 72 **RAM blocks:** 397 **Mem. bits:** 2 M	(8,8)
Arria 10 GX 1150	2 min	3 min	4 min	8.5 hrs	**ALM:** 427 k **DSP:** 1516 **RAM blocks:** 2713 **Mem. bits:** 55.5 M	**ALM:** 129 k **DSP:** 300 **RAM blocks:** 1091 **Mem. bits:** 16 M	(16,32)

as the building block of all the layers. This leads to have limited options to increase the level of parallelism that can be exploited with a given network algorithm. Relaxing this limitation, (i.e. manually designing kernels based on each layers' computation flow) could lead to better resources consumption and higher throughput at the expense of losing the scalability that is needed for automation of this process. The maximum operating frequency (f_{max}) varies for different FPGAs. Indeed it depends on the underlying technology and FPGA family. Intel OpenCL compiler (synthesizer) automatically adjust PLLs on the board to use the maximum frequency for the kernels. It is of interest that the operating frequency of the kernels supporting AlexNet and VGG-16 were the same as they had essentially the same critical path, even though VGG-16 is a lot more complex. The larger complexity of VGG was handled by synthesizing a core (i.e. Figure 3.6) that executes a greater number of cycles if the number of layers in the network increases.

Table 3.3 Comparison to the existing works AlexNet with hardware options $(N_i, N_l) = (16, 32)$

	AlexNet [67]	**AlexNet [68]**	**AlexNet [45]**	**AlexNet [66]**	**AlexNet [This work]***
FPGA	Virtex-7 VX485T	Stratix-V GXA7	Zynq 7045	Stratix-V GX-D8	Arria 10 GX1150
Synthesis Method	C/C++	RTL	C/C++	OpenCL	OpenCL
Frequeny (MHz)	100	100	125	-	199
Logic Utilization	186K (61%)	121K (52%)	-	120K (17%)	129K (30%)
DSP Utilization	2,240 (80%)	256 (100%)	897 (99.5%)	665 (34%)	300 (20%)
Latency (ms)	21.61	12.75	8.22	20.1	18.24
Precision (bits)	32 float	8-16 fixed	16 fixed	8-16 fixed	8 fixed
Performance (GOp/s)	61.62	114.5	161.98	72.4	80.04

*batch size = 1

Table 3.4 Comparison to the existing works VGG-16 with hardware options $(N_i, N_l) = (16, 32)$

	VGG-16 [85]	**VGG-16 [47]**	**VGG-16 [45]**	**VGG-16 [66]**	**VGG-16 [This work]***
FPGA	Zynq 7045	Arria 10 GX1150	Zynq 7045	Stratix-V GX-D8	Arria 10 GX1150
Synthesis Method	-	RTL	C/C++	OpenCL	OpenCL
Frequeny (MHz)	150	150	125	120	199
Logic Utilization	182 (83.5%)	161K (38%)	-	-	129K (30%)
DSP Utilization	780 (89.2%)	1518 (100%)	855 (95%)	-	300 (20%)
Latency (ms)	-	47.97	249.5	262.9	205
Precision (bits)	16 fixed	8-16 fixed	16 fixed	8-16 fixed	8 fixed
Performance (GOp/s)	136.91	645.25	161.98	117.8	151.7

*batch size = 1

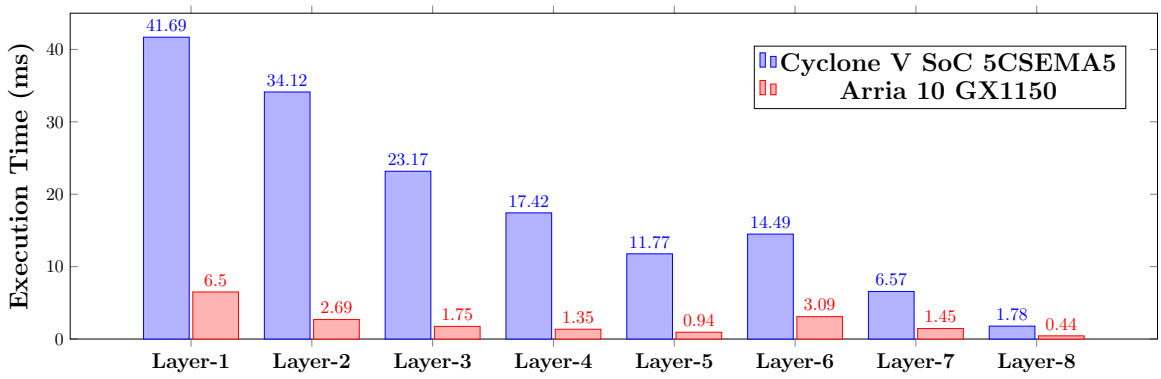


Figure 3.8: Detailed breakdown of execution time for each layer of AlexNet. A Layer here means execution of one round of pipelined kernels as shown in Figure 3.6. In case of AlexNet Layer-1 to Layer-5 are combination of memory read/write, convolution and pooling kernels while Layer-5 to Layer-7 are combination of memory read/write and fully connected kernels

Table 3.2 gives more details regarding the design space exploration algorithms which are coupled with synthesis tool. All three algorithms use the resource utilization *estimation* of the synthesizer to fit the design on the FPGA. This is important as the time consumed for design space exploration is normally under 5 minutes, while the synthesis time for larger FPGAs such as the Arria 10, can be close to 10 hours.

Experimenting with various DSE algorithms confirms that smart algorithms (such as reinforcement learning and hill climbing) can provide advantages for design space exploration when compared to brute-force enumeration. The first goal of suggesting the RL-DSE and HC-DSE methods is to demonstrate that it is possible to further decrease the exploration time by using some better search methods. Analysing the execution times shows that the reinforcement learning algorithm is almost 25 percent and the hill climbing is 50 percent faster than the brute-force algorithm when optimizing the design for the Arria 10 FPGA. Note that these exploration times are significantly less than synthesis time, since we use estimation of resource usage provided by the synthesizer. On the other hand, performing full synthesis as part of the exploration process is not advisable because the exploration time would take weeks. Since HC-DSE follows the gradient of resource usage, it is always faster than BF-DSE. This method is the best when the design-space is convex. However, if the search space is not convex (e.g. having several local maximums) HC-DSE might choose a wrong solution. In contrast, it is less probable for RL-DSE to be trapped in a local maximum due to the random nature of the reinforcement learning algorithm. Moreover, the RL-DSE algorithm would be more valuable if it could be exploited in conjunction to the reinforcement

learning quantization algorithms such as ReLeQ [71].

The goal of considering various DSE algorithms in our work is to provide a versatile tool for the user in different conditions and is not limited to a specific case. We included the brute-force algorithm in CNN2Gate in order to guarantee a successful exploration for small design-spaces. However, presuming that the design space is small is not always a correct assumption. For large convex design-spaces, we added the hill climbing algorithm to find the best solution which works significantly faster than brute-force. For non-convex and large design-spaces, reinforcement learning was found to work well [86]. In Table3.2 the design-space is small. This is why the difference between the execution time between various exploration algorithms is small. However, it was shown that HC-DSE can be twice faster than brute force when optimizing the design for the Arria 10 FPGA.

There are other model-based design space exploration algorithms that are dedicated to a specific implementation of a library of primitives. For instance, in [69], the authors proposed a *performance model* for their design, and they can predict the performance of the hardware implementation based on the resource requirements. The advantage of our proposed DSE algorithms is that our algorithms are model agnostic. This means that the CNN2Gate framework tunes the parallelism parameters of the design and queries the performance feedback directly from the synthesizer.

We tried CNN2Gate on three platforms. The first one is a very small Cyclone V device with 15k Adaptive Logic Modules (ALMs) and 83 DSPs. The fitter could not fit either ALexNet or VGG on this device. Clearly, the minimum space required for fitting this design on FPGA is fairly large due the complexity of the control logic. CNN2Gate did not experience any difficulty fitting the design on bigger FPGAs as demonstrated in Table 3.2. Resource utilization and hardware options (N_i, N_l) are also provided, which correspond to the execution times shown in Table 3.1.

CNN2Gate resource consumption is very similar for AlexNet and VGG-16. In case of identical hardware options, CNN2Gate's synthesized core is going to be nearly identical for all CNN architecture as shown in Figure 3.6. The only difference is the size of internal buffers to allocate the data in the computation flow. More quantitatively, in our implementation, VGG-16 uses 8% more of the Arria 10 FPGA block RAMs in comparison to what is shown in Table 3.2 for AlexNet.

Revisiting Figure 3.6, pipelined kernels are capable of reading data from global memory and process the convolution and pooling kernel at the same time. In addition, for fully connected layers in CNNs, the convolution kernel acts as the main data process unit and the pooling kernel is configured as a pass-through. Considering this hardware configuration, we can

merge convolution and pooling layers as one layer. In the case of AlexNet, this leads to five fused convolution/pooling and three fully-connected layers. Figure 3.8 reports the detailed execution time of these six layers including memory read and write kernels for each layer. Thus, as the algorithm proceed through the layers, dimensions of the data (features) reduced and the execution time is decreased. Note that in this figure, Layer 1 to Layer 5 are fused layers comprising a convolution, a pooling, and a ReLu layers. Also note that although the number of parameters in convolutional layers are less in fully connected layers, the memory consumption is far greater than with fully connected layer. Therefore, the performance of the first convolutional layers can be affected in a system accessing external memory (RAM).

Table 3.3 shows a detailed comparison of CNN2Gate to other existing works for AlexNet. CNN2Gate is faster than [66, 67] in terms of latency and throughput. However, CNN2Gate uses more FPGA resources than [66]. To make a fair comparison, we can measure relative performance density per DSP or ALMs. In this case, CNN2Gate performance density (GOp/s/DSP) is higher (0.266) when compared to 0.234 for [66]. There are other designs such as [45, 68] which are faster than our design in terms of pure performance. Nevertheless, the CNN2Gate method outlined above significantly improves on existing methods as it more scalable and automatable than the methods presented in [45, 68], which are limited in these regards as they require human intervention in order to reach high levels of performance.

Second, the design entry of [45, 68] are C and RTL respectively, while our designs were automatically synthesized from ONNX using OpenCL. Thus, not surprisingly, our work does not achieve the maximum reported performance. This is partly due to the use of ONNX as a starting point, and trying to keep the algorithm scalable for either large and small FPGAs. This imposes some limitations such as the maximum number of utilized CONV units per layer. There are also other latency reports in the literature such as [44]. However, those latency reports are measured with the favorable batch size (e.g. 16). Increasing batch size can make more parallelism available to the algorithm that can lead to higher throughput. Thus, for clarity, we limited the comparisons in Table 3.3 and 3.4 to batch size = 1.

Table 3.4 shows a detailed comparison of CNN2Gate to other existing works for VGG-16. It is observable that CNN2Gate is performing better for larger neural networks such as VGG. CNN2Gate achieves 18 % lower latency than [45], despite the fact that CNN2Gate is using fewer DSPs. While for AlexNet, [45] was more than 50 % faster than CNN2Gate. Finally, for VGG-16, we did find some hand tailored RTL custom designs such as [47] that are faster than CNN2Gate.

3.7 Discussions and Generalization

In this section, we explore CNN2Gate from the point of view of its software architecture. We also discuss some limitations of CNN2Gate that is caused by the library of primitives over which it is built. Finally, generalized forms of design-space search algorithms are discussed, which can be very insightful for future designs.

3.7.1 CNN2Gate as an External Controller with a Synthesizer in the Loop

In order to better understand the design of CNN2Gate, it is helpful to see this framework in the context of a controller. CNN2Gate provides two sets of actions as a controller:

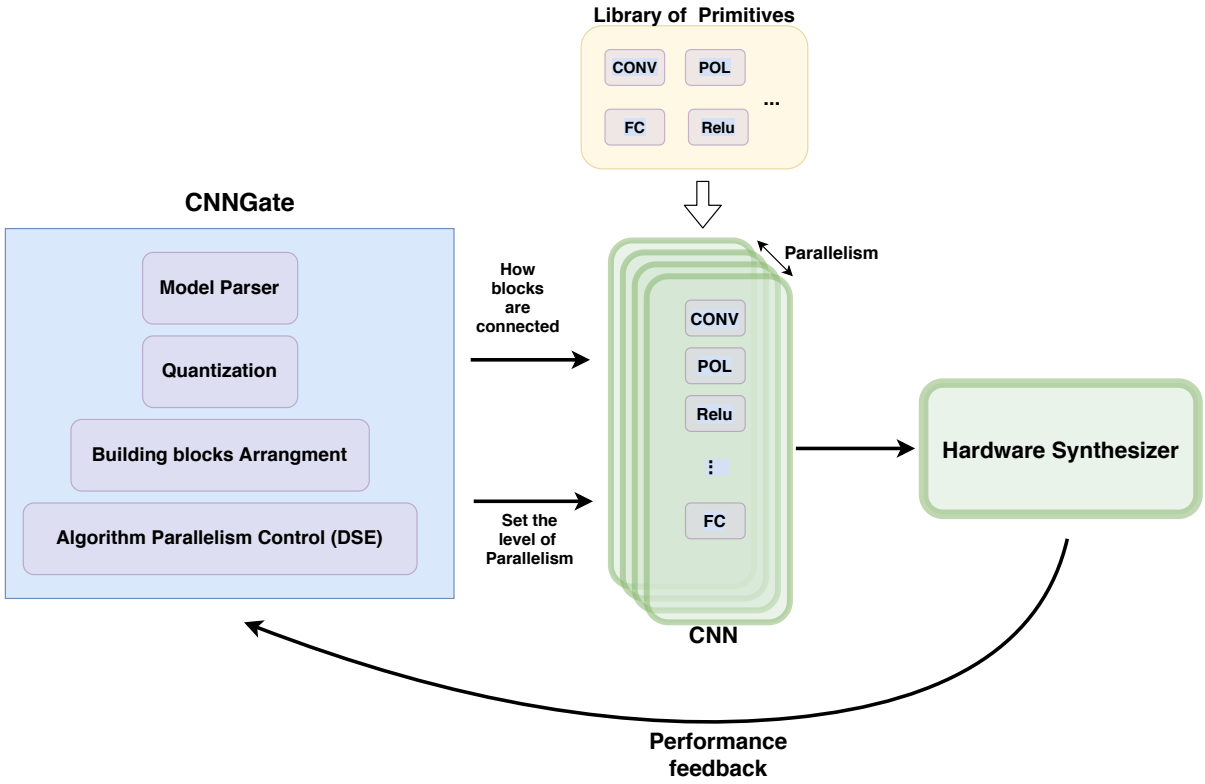


Figure 3.9: CNN2Gate as an external controller

1. **Arranging the basic hardware building blocks or library of primitives:**

   CNN2Gate provides the framework in which the primitive blocks are connected to form the specific architecture of a convolutional neural network. This architecture is based on the model that is parsed from an ONNX representation of the neural network. As shown in Figure 3.9 CNN2Gate chooses and connects the building blocks of a convolutional neural network from a library of primitives. This library of primitives can be designed by the user or can be adapted from other existing open-source designs. We assume that the level of parallelism is controllable in the library of primitives.

2. **Providing parallelism control for algorithm or design space exploration:**

   After providing the layout of building blocks, CNN2Gate makes several queries to the hardware synthesizer and determines the best parameters particularly regarding the level of parallelism in order to get the best performance or performance/accuracy/cost trade-off of the hardware implementation. The performance characteristics of the hardware architecture can be defined by the user as the total logic utilization, throughput, or combination of them.

In our case, we adapted the library of primitives from PipeCNN [44] . The three DSE algorithm that we are introduced in the previous sections are the control algorithms that work in conjunction with the synthesizer to fit the design to the desired FPGA target.

3.7.2 Ablation Study of the Basic Hardware Building Blocks

Since CNN2Gate's primitive libraries are adapted from [44], it inherits the limitations of PipeCNN. Nevertheless, the CNN2Gate is flexible and can be easily adapted to other primitive libraries. Being in the form of an external controller, CNN2Gate offers the possibility of continuous integration to new DSE algorithms and primitive hardware libraries. For instance, a more recent version of the PipeCNN primitive library were introduced to support RESNET [87]. In this case, the CNN2Gate parser block can be changed to support the new architecture. Moreover, the design-space algorithms can stay the same.

The methods that CNN2Gate uses are flexible enough to be adapted for the implementation of Recurrent Neural Networks (RNNs). This requires the user to provide the primitive libraries of the RNN. When the library of primitive changes completely, the number of parameters that control the parallelism in the algorithm might change. This requires having a generic N-dimensional design space exploration. In the next Section, possible generalizations of design-space algorithms are explained.

3.7.3 Generalizations of Design Space Exploration Algorithms

In Section 3.5.3, some algorithms are introduced to explore the two-dimensional design space. These algorithms can be easily generalized to N-dimensional explorations. The following sections demonstrate this generalization.

N-dimensional Hill Climbing Design Space Exploration

The hill climbing algorithm is widely used in artificial intelligence to find local maximums. The relative simplicity of this algorithm makes it the first choice for our exploration algorithms. The hill climbing algorithm will find the global maximum if the search space is convex.

Algorithm 4: Generalized Hill climbing Algorithm

```
Input  : H_0 = (x_1_0, x_2_0 ... x_n_s0)
Output: H_best
H_current := H_start
compute: O(H_current)
while true do
    H_next := null
    F_avg(next) := -inf
    for i : Neighbours(H_current) do
        if (H_i satisfies boundary limitations then
            if O(H_i) > O(H_next) then
                O(H_next) = O(H_i)
                H_next = H_i
    if O(H_next) ⩽ O(H_current) then
        H_best = H_current
        return : H_best
```

As shown in Algorithm 4, A hill climbing optimizer starts from an initial point $H_0 = (x_{1_0}, x_{2_0}...x_{n_0})$. This initial point can be chosen randomly in the search space. In each step, the optimizer visits all the neighbours of the current location and examine the objective function O, if the optimizer finds a better choice among the neighbours , it updates its position to the better choice. This procedure continues iteratively until the hill climber reaches the local maximum and cannot find a better point in its neighborhood. Note that the objective function O can be defined as the total logic utilization, throughput or combination of them.

N-dimensional Reinforcement Learning Design Space Exploration

As shown in Figure 3.10, a reinforcement learning software agent takes actions in the environment in order to maximize the reward. In the context of CNN2Gate, an RL agent is used to obtain the optimal control to choose the best hardware options for the model in the synthesis process. Considering CNN2Gate as an external controller (i.e. Figure 3.9) fits very well with the paradigm of a reinforcement learning agent. In this context, CNN2Gate is the *'Agent'* that explores the available synthesis options of the convolutional neural networks (i.e. *'Environment'*). Controlling the various characteristics of a neural network using reinforcement learning has appeared recently in the literature. For instance, in [70], an RL agent is used to find the best quantization level for each layer.

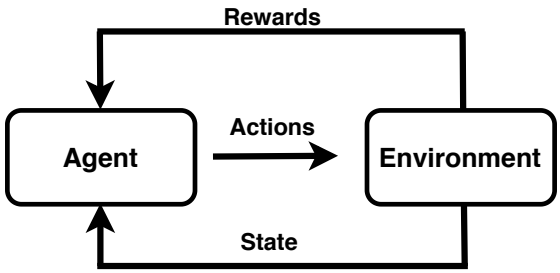


Figure 3.10: Reinforcement Learning: A software agent takes actions in the environment in order to maximize the reward

To set up the RL agent to explore the N-dimensional design-space we need to take into consideration the following steps:

1. **Defining the states:** In the context of a hardware optimizer, states can be defined as a set of all possible hardware options. Particularly, states are representing the whole design-space.

2. **Defining the actions:** Actions are the act of moving from one state to another. In the case of N-dimension, actions can be defined as the unit vector of each dimension.

3. **Forming a reward function:** Executing an Action a in the state s provides the agent a numerical reward score r. The agent tries to maximize this reward. In the context of this article, the reward can be throughput.

4. **Considering a living penalty or discount factor for the agent:** The agent should not have an infinite amount of time to explore the environment. The longer the agent

stays in the environment, the less reward it should gets. This trains the agent to find the best hardware option as fast as possible.

Having defined all the previous steps, we are able to train our agent to find the best hardware option using common reinforcement learning techniques such as Q-Learning [88].

3.8 Conclusion

This paper described the design and implementation of a general framework for developing convolutional neural networks on FPGAs. This framework takes the form of a Python library that can be integrated with a wide range of popular machine learning frameworks. CNN2Gate makes it easy for machine learning developers to program and use FPGAs to perform inference. CNN2Gate exploits the OpenCL synthesis workflow for FPGAs offered by commercial vendors. CNN2Gate is capable of parsing the data-flow of CNN models expressed in ONNX. This framework also has an integrated design space exploration tool that helps developers finding the best hardware option for synthesizing a given CNN model on an FPGA. CNN2Gate achieves a classification latency of 205 ms for VGG-16 and 18 ms for AlexNet on an Intel Arria 10 FPGA. These results are excellent considering that they were obtained by an automated design space exploration and synthesis process which is not relying on expert's low-level hardware knowledge.

CHAPTER 4 ARTICLE 2 : EFFICIENT DESIGN SPACE EXPLORATION OF OPENCL KERNELS FOR FPGA TARGETS USING BLACK BOX OPTIMIZATION

This chapter is a reproduction of an article published in IEEE Access on October 4, 2021.

- Ghaffari, A. and Savaria, Y., 2021. Efficient Design Space Exploration of OpenCL Kernels for FPGA Targets Using Black Box Optimization. IEEE Access, 9, doi: 10.1109/ACCESS.2021.3117560.

4.1 Abstract

Nowadays, many industries are in favor of using intelligent design space exploration as opposed to brute-force analysis. In many applications, the design space is defined by multiple variables and their interactions. Although brute-force analysis is very simple, it is rarely scalable when the number of variables in the system increases. With the rising complexity of hardware designs, more intelligent approaches are needed to explore the design options. This paper proposes using smart meta-heuristic search algorithms such as Grey Wolf Optimization (GWO) in conjunction with Bayesian Optimization (BO) to solve this problem. We show that we can further reduce the design effort using a surrogate model that is created based on a novel hybrid GWO-BO method. The surrogate model is a useful abstraction to detect functional and physical inter-dependencies in the system in order to accurately predict its performance (e.g. throughput or latency). We evaluate our methodology and show that it can produce competitive results in order to find the best design variables that maximize the performance of the system. Finally, we compare our results with previous statistical and heuristic methods proposed in the literature and find that the proposed GWO-BO method always performs better than the other considered methods.

4.2 Introduction

Efficient hardware is a ubiquitous requirement in industrial and scientific problems. To achieve the best efficiency, designers need to take into account the effect of many design variables. Exploring all possible combinations of variables is often impractical. On the other hand, the interaction between variables and their related trade-offs are often unknown or differ from one design to another. More specifically, in hardware design, synthesizing a circuit to

evaluate one set of variables can take hours, while designers must explore tens of thousands of design options to find a high quality solution. Moreover, since there is no gradient defined for most hardware variables (e.g. memory block size, number of parallel data paths, etc), gradient-based optimization is not a viable option to explore the design space.

In such a complex multi-dimensional optimization problem, human expert choices can be inaccurate or biased. Furthermore, understanding and estimating a multi-dimensional, non-linear non-convex objective function is difficult for most human experts, and it is easy to miss optimal design solutions.

In particular, with the rise of High-Level Synthesis (HLS), it is possible to design hardware solutions using high-level languages such as C/C++ and OpenCL. While HLS makes the design process easier and faster, understanding the design space is harder. For instance, when using OpenCL, the increasing number of threads and number of SIMD (Single Instruction Multiple Data) instructions per thread both seem to increase performance. However, finding the optimum choices for threads and SIMD elements per thread to achieve the best throughput/area usage is not obvious.

It is commonly known that using meta-heuristics such as Genetic Algorithm (GA) [89] or derivative-free optimizations [90] can help with such design optimization problems. As we will explore in this paper, the issue with meta-heuristic algorithms is that they do not provide any understanding or estimation of the objective function. On the other hand, it is also well known that statistical models such as Bayesian Optimization (BO) [23] and LASSO (Least Absolute Shrinkage and Selection Operator) [91] can provide some understanding of the model behavior. However, statistical models such as BO are extremely sensitive to the data samples provided to them. This means that if the samples are not correctly drawn from the environment, the surrogate model might not behave like the actual model.

In this paper, we propose algorithms that perform design space exploration for OpenCL kernels. By using a simple Hill climbing algorithm, we show that a hill climber often reaches a local maximum (or minimum), thereby demonstrating that the design space is often non-convex. Next, we propose Grey Wolf Optimization (GWO) as a meta-heuristic to explore the design space. Then, Bayesian Optimization (BO) with random samples from the environment is proposed to create a surrogate model of the system. Finally, by creating a hybrid optimizer (GWO-BO), we show that the quality of the samples explored by the wolves in the GWO algorithm is better than random samples in order to obtain a surrogate model that matches the maximum performance achievable by the actual physical design.

This work makes the following contributions:

- Providing a framework to optimize area×throughput OpenCL kernels in HLS.
- Proposing that Grey Wolf Optimization (GWO) is a competitive method to perform design space exploration.
- Building surrogate models that estimate the behavior of OpenCL kernels using a Bayesian Optimizer.
- Proposing a hybrid (GWO-BO) optimizer based on re-using the points explored by the GWO algorithm and feeding them to BO to create a surrogate model for HLS OpenCL kernels that systematically provide results better than both GWO and BO methods.

To the best of our knowledge **1)** this is the first time that GWO is proposed for hardware design space exploration and **2)** this is the first time that the explored variables of a meta-heuristic algorithm are re-used to build a more accurate surrogate model (i.e Hybrid GWO-BO method). The hybrid method enables benefiting from both algorithms without repeating expensive synthesis trials.

The rest of this paper is organized as follows: In Section.4.3, we review the literature related to this topic. In Section.4.4, we introduce three design space exploration methods as well as some background information regarding the algorithms and benchmarks that are used in this paper. In Section.4.5, we show the results from each algorithm and compare them with the benchmark. In Section.4.6, we study multi-objective optimization of throughput and area utilization, as well as Pareto-efficiency of the design variables. Finally, we compare our results with those obtained with previously proposed methods.

4.3 Related Works

Recently, intelligent design space exploration has attracted much attention among hardware designers and scientists. Hardware design space exploration can be treated as a black box optimization problem because first, evaluating the objective function is expensive, and second, the properties of the objective function such as derivatives and convexity are unknown. For years, brute-force analysis for small design spaces and meta-heuristic algorithms for larger design spaces were commonly used by designers to perform the exploration. For instance, meta-heuristic optimization algorithms such as the Genetic Algorithm [92] or Simulated Annealing [93] are typically used to explore large design spaces. The main problem with meta-heuristic methods is that, although they are very effective in finding optimum solutions, they cannot provide a suitable overall estimation of the system behavior (i.e. system's surrogate

model). Abstract estimation of a system's behavior is immensely important when trying to understand the inter-dependencies of the design variables.

On the other hand, BO [21] is an interesting approach for design space exploration because it is a form of incremental learning that provides an estimation of the objective function based on the previously observed samples. This is a very powerful strategy for finding the extremum of the objective functions that are not easy to evaluate. Furthermore, BO can estimate a *surrogate model* of the system that helps to explain the inter-dependencies of design variables. For instance, in [22], the authors suggested a framework for design space exploration for C/C++ High-Level Synthesis that uses BO. They have shown that BO outperforms the traditional search method in terms of latency and resource usage in the FPGA targets. Similarly, in [23], the authors used BO to optimize hardware accelerators for deep neural networks. They reported that the BO method could help to simulate the accuracy of the deep neural network and energy efficiency of the accelerator hardware. In [20], BO was applied to tune directives to achieve minimum latency.

Although the approach we are presenting also uses BO to perform design space exploration, there are two distinct features that differentiate our work from [22, 23]. First, our work is based on High-Level Synthesis for OpenCL kernels. Design space exploration for OpenCL kernel poses new challenges as the fine-grained thread-level parallelism must be controlled for the FPGA device. Moreover, as opposed to the above mentioned work, we did not discard the meta-heuristic methods. Our work also explores using a new meta-heuristic method, Grey Wolf Optimization [52], that is suitable for design space exploration. Furthermore, we combined GWO and BO methods to obtain the best results.

Inspired by traditional design space exploration using meta-heuristics, we believe there is still room for more research using meta-heuristic methods and their combination with BO. For instance, the recently suggested Grey Wolf Optimization (GWO) [52] exhibits competitive performance in design space exploration applications. In this research, we show that samples that have been chosen by the GWO algorithm can be very suitable candidates for constructing a surrogate model that estimates the minimum latency.

Reinforcement learning [28] has also been shown effective for design space exploration. Similar to Bayesian optimization, reinforcement learning is also an incremental learning method that directs the search agent toward the optimum solution by analyzing the previously observed samples using a Q-table or neural network. In [29], the authors used reinforcement learning to explore the optimization of deep neural networks on the ARM-Cortex-A CPUs. Likewise, in [30], the authors used a time-limited reinforcement learning [31] to execute design space exploration for deeply pipelined OpenCL kernels of the convolutional neural networks.

Finally, there are other machine learning-based exploration techniques, such as random forest [18] for HLS design space exploration, and Markov decision process [94] for exploration of multi-processor platforms that have appeared in the literature. Among those researches, Bayesian and meta-heuristic methods still exhibit more competitive exploration performance for large design spaces.

4.4 Methodology

The design space exploration problem can be considered as a Black Box Optimization (BBO) [90] problem. Typically, a black-box optimizer is a type of optimizer that assumes the objective function is unknown. Furthermore, there is no assumption of any form of continuity, convexity, and smoothness of the objective function. The black-box methods that are used in this work are derivative-free, which is beneficial because, for most hardware design variables, the derivative does not exist or is hard to compute. Historically, meta-heuristic methods such as genetic algorithm [95] are well-known methods to solve the black box optimization problem. Although meta-heuristics are fast at finding the optimum solution, they do not provide any abstract modeling (i.e. surrogate model) of the black box objective function. Moreover, they are not guaranteed to converge.

Figure 4.1 summarizes the framework that is discussed in this paper. A design space exploration framework can be defined as a black box optimizer that receives resource utilization and throughput performance from either a physical design or a synthesizer tool. In this scheme, the design space explorer may save the samples that it receives from the environment in order to create a surrogate model of the objective space. Furthermore, the black box optimizer provides the optimum design variables to the OpenCL kernels.

In the following section, first, the hill climbing method and its inefficiency for non-convex design spaces is reviewed. Second, Grey Wolf Optimization is proposed as a heuristic method for design space exploration and third, Bayesian optimization is reviewed as a tool to obtain the surrogate model of the objective function. Fourth, an ensemble method combining GWO and BO is proposed. Finally, after presenting the OpenCL benchmark suite that we used to evaluate our results, it is shown that the proposed GWO-BO method achieves the best results on that benchmark suite.

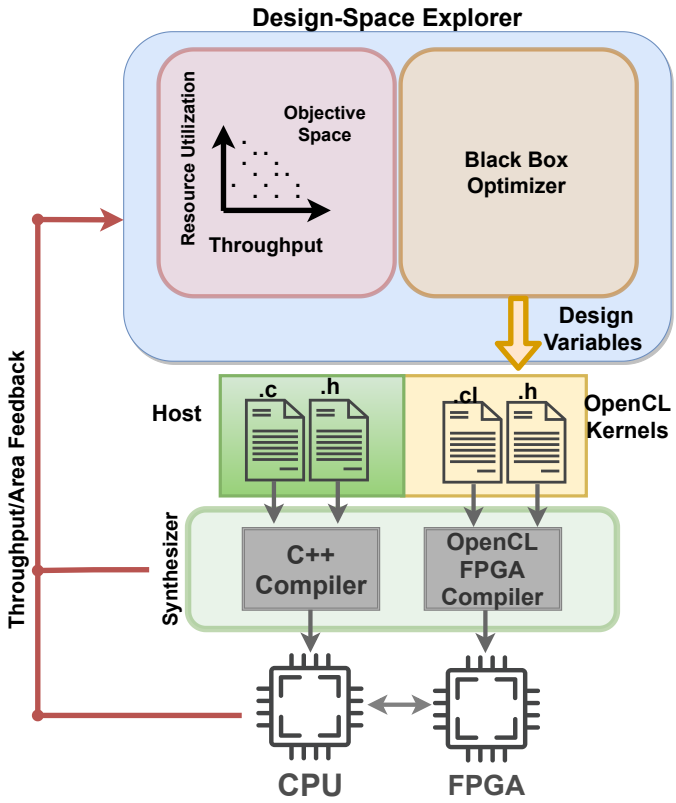


Figure 4.1: A design space exploration framework which consist of a black-box optimizer and a synthesizer in the feedback loop

---

Algorithm 5: Generalized Hill climbing Algorithm

Input : $\mathbf{x_0}$: Initial variables.

Output: $\mathbf{x}^{\mathbf{best}}$: The best set of variables found by Hill climbing

$\mathbf{x}^{\mathbf{current}} := \mathbf{x_0}$

compute: $O(\mathbf{x}^{\mathbf{current}})$

while *true* **do**

  $\mathbf{x}^{\mathbf{next}} := null$

  $O(\mathbf{x}^{\mathbf{next}}) := -inf$

  for i : ***Neighbours***$(\mathbf{x}^{\mathbf{current}})$ **do**

    if *($\mathbf{x}_i$ satisfies boundary limitations)* **then**

      if $O(\mathbf{x}_i) > O(\mathbf{x}^{\mathbf{next}})$ **then**

        $O(\mathbf{x}^{\mathbf{next}}) = O(\mathbf{x}_i)$

        $\mathbf{x}^{\mathbf{next}} = \mathbf{x}_i$

  if $O(\mathbf{x}^{\mathbf{next}}) \leqslant O(\mathbf{x}_i)$ **then**

    $\mathbf{x}^{\mathbf{best}} = \mathbf{x}^{\mathbf{current}}$

    return : $\mathbf{x}^{\mathbf{best}}$

4.4.1 Hill Climbing as a Measure of Non-Convexity

Hill climbing [89] is an iterative numerical algorithm that starts from an arbitrary location in the design space and attempts to find a better solution by moving incrementally toward the optimum solution. This incremental process continues until there is no further improvement in the objective function. Although the Hill climbing algorithm is very efficient at finding local optimums, it usually fails to find the global optimum in a non-convex environment. Given that hill climbing is not a good strategy for non-convex design spaces, we use it in this paper as a measure of the non-convexity of a design space. Note that a hill climber might fail to find the global optimum as it gets stuck in a local optimum, this means the design space is non-convex.

As shown in Algorithm 5, a hill climber algorithm starts from an initial variable vector $\mathbf{x_0}$, which is chosen randomly in the design space. In every iteration, the optimizer visits all the neighbors of the current variable and examines the black-box objective function O, and if the optimizer finds a better variable set among the neighbors, it updates its position to it. This procedure continues until the hill climber reaches the local maximum and cannot find better design variables in its neighborhood.

4.4.2 Grey Wolf Optimization

According to the No Free Lunch theorem [96], it can be proven that no meta-heuristic can solve all optimization problems. To address the ever-increasing complexity of design space exploration with many problems of interest, there is a need to propose and explore new meta-heuristic strategies that provide superior performance in a given design space. GWO [52] is a meta-heuristic algorithm that is inspired by the hierarchy and hunting strategy of grey wolves and exhibits very competitive performance in design space exploration problems. In GWO, a fixed number of search agents (i.e. wolves) explore and exploit the search environment in order to find the optimum solution (i.e. prey). Furthermore, the search agents share information about the search space and assist each other to avoid local optimums.

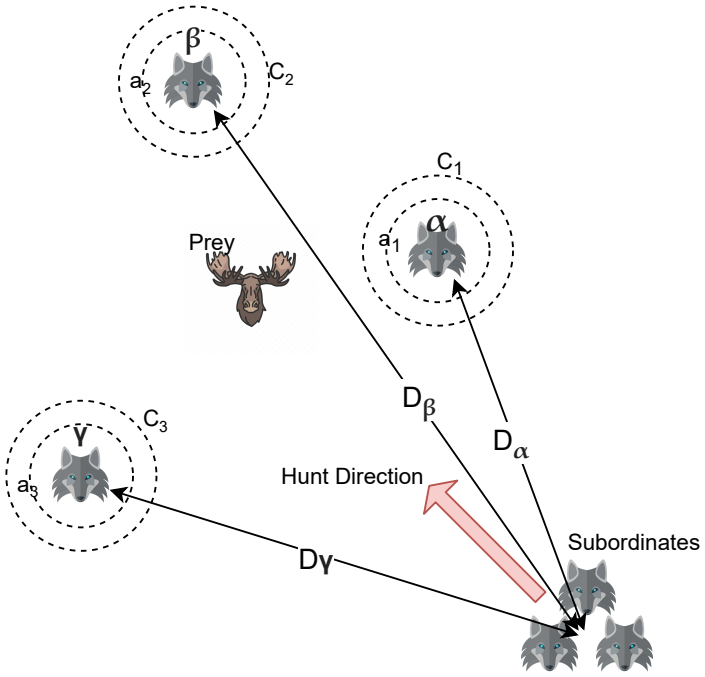


Figure 4.2: Position update methodology in Grey Wolf Optimization

---

Algorithm 6: Grey Wolf Optimization Algorithm

Input : Initialize number of wolves

Input : Initialize variables $\hat{\mathbf{a}}$, $\mathbf{a}$ and $\mathbf{c}$ (eq (4.2))

Output: $\mathbf{x}_t^{\alpha}$

Compute: fitness of each wolf with objective function: $O(\mathbf{x_t})$

$\mathbf{x}_t^{\alpha}$: the best search agent (wolf)

$\mathbf{x}_t^{\beta}$: the second best search agent (wolf)

$\mathbf{x}_t^{\gamma}$: the third best search agent (wolf)

while $t <$ *Max Iterations* **do**

- **for** *wolf : **wolf population*** **do**
  - Update wolf positions $\mathbf{x}_t$ using eq (4.5)
  - **if** *($\mathbf{x}_t$ does **not** satisfy boundary limitations)* **then**
    - Reinitialize the position randomly,
- **Update:** $\hat{\mathbf{a}}$, $\mathbf{a}$ and $\mathbf{c}$
- **Compute:** fitness of each wolf with objective function: $O(\mathbf{x}_t)$
- **Update:** best search agents: $\mathbf{x}_t^{\alpha}$, $\mathbf{x}_t^{\beta}$ and $\mathbf{x}_t^{\gamma}$

In nature, grey wolves encircle their prey. This phenomenon can be mathematically formulated as:

$$\begin{aligned} \mathbf{d} &= |\mathbf{c}.\mathbf{x}_t^{\mathbf{prey}} - \mathbf{x}_t| \\ \mathbf{x}_{t+1} &= \mathbf{x}_t^{\mathbf{prey}} - \mathbf{a}.\mathbf{d} \end{aligned} \tag{4.1}$$

Where $\mathbf{d}$ is the relative distances of the wolves to the prey, $\mathbf{x}^{\mathbf{prey}}$ is the position of the prey, $\mathbf{x}_t$ is the position of wolves on iteration t, and $\mathbf{a}$ and $\mathbf{c}$ are coefficient vectors calculated by:

$$\begin{aligned} \mathbf{a} &= 2\hat{\mathbf{a}}.\mathbf{r_1} - \hat{\mathbf{a}} \\ \mathbf{c} &= 2.\mathbf{r_2} \end{aligned} \tag{4.2}$$

Where $\mathbf{r_1}$ and $\mathbf{r_2}$ are uniformly distributed random variables between $[0, 1]$ and elements of the vector $\hat{\mathbf{a}}$ are linearly decreased from 2 to 0 as the algorithm iterates. Note that $\hat{\mathbf{a}}$ controls the exploration and exploitation of the algorithm. When $\hat{\mathbf{a}}$ is 2, the GWO is in the exploration mode and as it goes toward 0, the algorithm enters the exploitation mode.

Inspired by the hierarchy of wolves, the search is guided by the three fittest search agents namely α , β and γ wolves. This means that the position of the prey is estimated by the position of the three fittest agents in each iteration. As shown in the following equations, we can model the process of encircling the prey using the estimated position of the prey by:

$$\begin{aligned} \mathbf{d}^\alpha &= |\mathbf{c}.\mathbf{x}_t^\alpha - \mathbf{x}_t| \\ \mathbf{d}^\beta &= |\mathbf{c}.\mathbf{x}_t^\beta - \mathbf{x}_t| \\ \mathbf{d}^\gamma &= |\mathbf{c}.\mathbf{x}_t^\gamma - \mathbf{x}_t| \end{aligned} \tag{4.3}$$

$$\begin{aligned} \mathbf{x}_1 &= \mathbf{x}_t^\alpha - \mathbf{a_1}.\mathbf{d}^\alpha \\ \mathbf{x}_2 &= \mathbf{x}_t^\beta - \mathbf{a_2}.\mathbf{d}^\beta \\ \mathbf{x}_3 &= \mathbf{x}_t^\gamma - \mathbf{a_3}.\mathbf{d}^\gamma \end{aligned} \tag{4.4}$$

and finally updating the position of other subordinate wolves by:

$$\mathbf{x}_{t+1} = round\left(\frac{\mathbf{x}_1 + \mathbf{x}_2 + \mathbf{x}_3}{3}\right) \tag{4.5}$$

Note that since our design space is discrete, the $\mathbf{x}_{t+1}$ is rounded to the nearest integer.

Figure 4.2 demonstrates the direction of the search based on the position of α , β, and γ wolves. Algorithm. 6 shows the GWO pseudo code utilizing eq (4.2) to (4.5) .

4.4.3 Bayesian Optimization

Bayesian optimization [21] is a powerful method to find the extremum of an objective function that is expensive to evaluate. For instance, in hardware design space exploration problems, it normally takes hours to synthesize an OpenCL code using high-level synthesis tools. Bayesian Optimization assumes that the *posterior* probability of an objective function O for a given evidence sample data is proportional to the likelihood of samples multiplied by *prior* probability of O.

Let us define $\mathbf{x}_i$ as the i'th sample and $O(\mathbf{x}_i)$ as the evaluation of the objective function for that sample. According to Bayes theorem, for a collection of t samples such as $C_{1:t} = \{\mathbf{x}_{1:t}, O(\mathbf{x}_{1:t})\}$:

$$P(O|C_{1:t}) \propto P(C_{1:t}|O) \times P(O) \tag{4.6}$$

where $P(O|C_{1:t})$ denotes the posterior distribution. Since the posterior distribution accumulates beliefs about the objective function, it can be represented as a *surrogate function* which estimates the objective function O. Note that in theory, Bayesian optimization assumes that the objective function is continuous. However, it is possible to estimate a surrogate function for a discrete objective function by fitting a continuous posterior function that passes through the discrete points of the objective function O. This idea is also explored in [24].

In practice, the sampling process is considered noisy with a Gaussian Distribution $\mathcal{N}(0, \sigma^2)$ where σ denotes the noise that is incorporated with the observation. Likewise, the surrogate function is also considered to be a Gaussian Process. This means the surrogate function f_s returns mean and variance as opposed to a normal function which returns a scalar value.

$$f_s(\mathbf{x}) = \mathcal{GP}(m(\mathbf{x}), \mathbf{K} = k(\mathbf{x}, \mathbf{x}'))) \tag{4.7}$$

Eq (4.7) shows that the $\mathcal{GP}$ is constructed of a mean vector m and a covariance matrix $\mathbf{K}$ comprising of jointly Gaussian distribution between the sample population $C_{1:t}$ and the next chosen sample $\mathbf{x}_{t+1}$. For this research, the squared exponential covariance kernel k is used to construct covariance matrix $\mathbf{K}$:

$$k(\mathbf{x_i}, \mathbf{x_j}) = \exp(-\frac{|\mathbf{x_i} - \mathbf{x_j}|^2}{2}) \tag{4.8}$$

Bayesian optimization is efficient in terms of exploration and exploitation since it incorporates prior belief using jointly Gaussian kernels to help with the acquisition direction. Moreover, the search is directed using the *maximum probability of improvement* ($\mathcal{PI}$) function:

$$\mathcal{PI}(\mathbf{x}) = P(O(\mathbf{x^{next}}) \geq O(\mathbf{x})) = \Phi\left(\frac{m(\mathbf{x}) - O(\mathbf{x^{next}})}{\sigma(\mathbf{x})}\right) \tag{4.9}$$

where $O(\mathbf{.})$ is the objective function sample, Φ is the cumulative Guassian distribution, and $m(\mathbf{x})$ and $\sigma(\mathbf{x})$ are the mean and variance of the samples respectively.

Algorithm 7: Bayesian Optimization Algorithm

for $t = 1, 2, 3...$ ***Max Iterations*** **do**
- **Step 1.** Find $\mathbf{x}_t$ by maximizing acquisition function $\mathcal{PI}$ in eq (4.9) over current sample collection $C_{1:t-1} = \{\mathbf{x}_{1:t-1}, O(\mathbf{x}_{1:t-1})\}$
- **Step 2.** Sample the objective function $O(\mathbf{x}_t)$
- **Step 3.** Augment the data to the sample collection $C_{1:t}$
- **Step 4.** Calculate new covariance matrix using eq (4.8)
- **Step 5.** Update $\mathcal{GP}$ using eq (4.7)

Algorithm 7 shows the pseudo-code of a BO method. For simplicity, it is divided into five steps. Step 1 is the guided search using the acquisition function. Step 2 and Step 3 show the noisy sampling and augmenting new samples to the sample population. Finally, Steps 4 and 5 show the Gaussian process update procedure.

After obtaining a surrogate model for objective function O (i.e surrogate function $f_s(\mathbf{x})$), we define the top-1 surrogate model suggestion as $\arg\max_{\mathbf{x}}(f_s(\mathbf{x}))$ and top-5 surrogate model suggestion as a set of $\mathbf{x}_i$ that produce the 5 largest local maximums of the surrogate function $f_s(.)$.

4.4.4 Hybrid GWO-BO

Assuming that the user who optimizes the objective function O wants to benefit from both GWO and BO algorithms, it is possible to combine those two algorithms. Thus, the user runs GWO on the dataset and passes the explored variables by the GWO search agents to BO to create a surrogate model. In this article, we show that GWO can provide qualified samples

for Bayesian optimization and it can decrease the number of acquisitions of the objective function. Moreover, by changing the parameter $\hat{\mathbf{a}}$ in the GWO algorithm, it is possible to tune the exploration/exploitation balance for search agents.

The hybrid GWO-BO method can achieve superior performance compared to both GWO and BO methods alone, while constructing a model based on the samples explored by GWO which are considerably fewer than the number of samples used by the conventional BO model.

Algorithm 8: Hybrid GWO-BO Algorithm

Step 1. Perform GWO algorithm
Step 2. Save the wolves positions and their objective samples at each step t in the sample collection: $C_{1:t} = \{\mathbf{x}_{1:t}, O(\mathbf{x}_{1:t-1})\}$ } GWO

for $t = 1, 2, 3...$ *in* ***(Sample collection)*** **do**
  Step 3. Calculate new covariance matrix using eq (4.8)
  Step 4. Update $\mathcal{GP}$ using eq (4.7) } BO

return $\arg\max(\mathcal{GP})$

Algorithm 8 specifies the steps performed to search the design space using the hybrid GWO-BO method. First GWO collects the samples from the environment and saves them in the sample collections. Then the sample collection is passed to the BO to build the surrogate model using the Gaussian Process.

Section 4.5.4 discusses in detail the results obtained with the hybrid GWO-BO method.

4.4.5 Justification of the Design Space Exploration Algorithms Choices

We chose the GWO algorithm as it gives the user fine control over exploration/exploitation trade-offs using parameter $\hat{\mathbf{a}}$ as it will be discussed in Section.4.5.2. Moreover, GWO can be easily combined with BO to create a surrogate model of the design space. BO is used merely for the purpose of surrogate modeling. The surrogate model can help us understand the interaction between the design space variables without performing expensive synthesis analysis. It will be shown in Section.4.6.3 that a Bayesian surrogate model can represent the behavior of the design space more efficiently than regression models such as the LASSO (Least Absolute Shrinkage and Selection Operator) model. Finally, we proposed a hybrid GWO-BO model that improves the respective weaknesses of the individual GWO and BO algorithms. A hybrid GWO-BO model benefits from surrogate modeling of the BO, while it produces results that are at least as good as GWO.

4.4.6 Benchmark Algorithms

In order to validate the results of our black box optimization tool implementing the GWO-BO method, we used an OpenCL benchmark suite for FPGA targets called Spector [16]. The benchmark consists of ten benchmark functions that are commonly used in FPGA designs. The benchmark functions are OpenCL implementation of 1) Breadth-First Search Dense (BFS Dense) 2) Breadth-First Search Sparse (BFS Sparse) 3) Discrete Cosine Transform (DCT) 4) Finite-Impulse Response (FIR) filter 5) Histogram 6) Matrix Multiplication 7) Merge-Sort 8) 3D Normal Estimation 9) Sobel Filter and 10) Sparse Matrix-Vector Multiplication (SPMV).

Some design space variables are shared among the designs and some are specific to each design. For instance, *work-items* is a generic design variable that is related to the number of parallel threads in an OpenCL application. Likewise, *work-group* indicates the number of work-items to be used without shared memory, *compute-units* indicates the number of kernels launched on the device and *SIMD* shows the level of data-parallelism in the kernels. It is also possible to consider HLS directives such as *unrolling* as a binary variable that accepts 1 or 0 to enable or disable the functionality.

On the other hand, some variables are specific to each design. For example, the number of histograms in the histogram benchmark or the number of filter coefficients in the FIR benchmark.

4.5 Results

This section, characterizes, compares and discusses the design space exploration algorithms mentioned in Section. 4.4.

4.5.1 Hill Climbing and Non-Convexity Analysis

Introducing an algorithm that can perform optimally in non-convex design spaces is essential for researchers in this domain. Exploring a convex design space is fairly simple. For instance, the Hill climbing algorithm is very effective at finding the global optimum of a convex design spaces. However, in practice, it is not possible to guarantee the convexity of the design space or at least the landscape of the design space is unknown. Due to the unknown design landscape, exploration algorithms must be able to explore non-convex design spaces.

In this section, the Hill climbing algorithm is used solely as a metric to detect the non-convexity of the design space. This means that if the Hill climbing algorithm cannot find

the global maximum, the design space is not convex. Note that observing failure of the Hill climbing algorithm to find the global optimum is a *sufficient* condition to confirm non-convexity. However, it is not a *necessary* condition. With this, if Hill climbing can find the global maximum, it is not guaranteed the design space is convex.

Table 4.1 shows the execution of the Hill climbing algorithm over the benchmark functions mentioned in Section. 4.4.6. According to the Hill climbing metric, all benchmark functions except the Breadth-First Search Sparse (BFS Sparse) are non-convex. For BFS Sparse, we cannot determine the convexity since the Hill climbing algorithm has successfully found its best variable choices.

As demonstrated in Table 4.1, Hill climbing completely fails in determining the best latency of Matrix Multiplication and Sobel Filter benchmarks. Besides analyzing convexity, this experiment shows that designers should not ignore some simple incremental search algorithms such as Hill climbing in design space exploration problems. Although it is not possible to know the convexity prior to performing the experiments, it is always good to check whether Hill climbing is suitable. However, the essence of having a surrogate model to understand and analyze the interaction between the design variables remains unanswered with the Hill climbing method.

Table 4.1 Hill climbing as a measure of non-convexity.

Benchmark Function	**Hill Climbing Best Latency (ms)**	**Benchmark Best Latency (ms) [16]**	**Convexity**
BFS Dense	4.5051	4.28.4	Non-convex
BFS Sparse	12.5151	12.5151	Not determined
DCT	2.6458	2.1916	Non-convex
FIR	0.0566	7.3×10^{-4}	Non-convex
Histogram	1.7829	1.5481	Non-convex
Mergesort	1.9555	1.9269	Non-convex
Matrix Multiplication	152.4176	34.1980	Non-convex
Normal Estimation	6.6465	6.2779	Non-convex
Sobel Filter	3.7522	1.7190	Non-convex
Sparse Matrix Vector Multiplication	0.1791	0.1661	Non-convex

4.5.2 Grey Wolf Optimization (GWO)

As shown in Section 4.5.1, Hill climbing can provide some initial understanding regarding the convexity of the design space and latency of the benchmark functions. However, it underperforms in more complicated benchmarks such as Matrix multiplication and Sobel Filter.

The main reason is that Hill climbing is deterministic in its search scope, which means that it follows the direction of the largest increment in the objective function by evaluating the search agent's neighbors. Furthermore, it is not possible to control the trade-offs between exploration and exploitation in Hill climbing.

In contrast, GWO provides very flexible control over the exploration/exploitation trade-off. More specifically, the variable $\hat{\mathbf{a}}$ in eq (4.2) controls this trade-off. For instance, when $\hat{\mathbf{a}}$ is close to 2, the search agents are in their ultimate exploration mode. As $\hat{\mathbf{a}}$ decreases from 2 to 1, the intensity of exploration decreases. When $\hat{\mathbf{a}}$ is 1, the search agents enter the exploitation mode and as $\hat{\mathbf{a}}$ decreases from 1 to 0, the exploitation intensifies. The variable $\hat{\mathbf{a}}$ is set to linearly decrease from 2 to 0 in the course of iterations to provide the best trade-off between exploration and exploitation.

Eq (4.5) uses a **round to nearest integer** to quantize the positions of the wolves since GWO is performed over a discrete search space. The rounding affects the exploitation in a way that if there is not a great expected improvement, the search agents tend to stay in their current position. This phenomenon will further decrease the number of iterations needed for design space exploration.

Figure 4.3 demonstrates the effect of $\hat{\mathbf{a}}$ on search agents' exploration and exploitation trade-off for the Sobel Filter benchmark. The red triangles are the explored points by the agents while the blue circles show the design space. The balanced exploration and exploitation (i.e. $\hat{\mathbf{a}}$ is linearly decreased from 2 to 0) can achieve better results in terms of finding the best throughput performance.

Table 4.2 shows the performance of the GWO for the benchmark functions. The design space exploration tests were conducted for 100 trials and the average and standard deviation of latency has been reported here. Since GWO is a meta-heuristic algorithm that produces different results for different experiments, it is necessary to conduct at least 100 tests in order to understand the convergence behavior of the GWO.

GWO outperforms Hill climbing and finds near-optimum results for all the benchmarks. Specifically, for Sobel Filter and Matrix Multiplication benchmarks, GWO finds acceptable latency which was not possible to be found using Hill climbing method.

Although GWO provides interesting results for design space exploration, it is not enough to create a surrogate model to study the interaction between variables. Finding a surrogate model is the focus of the next sections.

Table 4.2 Design space exploration using Grey Wolf Optimization for 100 trials.

Benchmark Function	Average Latency (ms)	Standard Deviation of Latency (ms)	Benchmark Latency (ms) [16]	Average Number of Explored Variable Sets
BFS Dense	4.335	0.056169	4.28.4	85
BFS Sparse	12.5151	0	12.5151	85
DCT	2.5827	0.15291	2.1916	35
FIR	1.9×10^{-3}	9.6×10^{-4}	7.3×10^{-4}	140
Histogram	1.6634	0.093946	1.5481	120
Mergesort	1.9456	0.013607	1.9269	230
Matrix Multiplication	**39.7234**	8.2397	34.1980	130
Normal Estimation	6.493	0.14783	6.2779	110
Sobel Filter	**1.7819**	0.065844	1.7190	200
Sparse Matrix Vector Multiplication	0.16798	0.0028675	0.1661	100

4.5.3 Bayesian Optimization (BO)

In BO, a surrogate model is constructed using statistical methods (as explained in 4.4.3) on a set of random samples drawn from the environment. The quality of the surrogate model is proportional to the number of samples and the distribution of the samples.

Table 4.3 Design space exploration using Bayesian Optimization for 100 trials.

Benchmark Function	Average Latency of Optimum Point (ms)	Standard Deviation of Optimum Point (ms)	Average Latency of Top-5 Model Sugges-tions (ms)	Standard Deviation of Top-5 Model Sugges-tions (ms)	Benchmark Latency (ms) [16]	Number of Random Samples Evaluated
BFS Dense	4.4632	0.1401	4.3889	0.0619	4.28.4	101
BFS Sparse	13.3025	0.5139	12.7031	0.3525	12.5151	101
DCT	5.4859	4.9634	2.9319	0.5295	2.1916	42
FIR	3×10^{-3}	1×10^{-3}	2.6×10^{-3}	4×10^{-4}	7.3×10^{-4}	234
Histogram	6.7245	8.5840	3.2551	4.8735	1.5481	179
Mergesort	2.4989	0.3899	2.0935	0.1882	1.9269	306
Matrix Multiplication	41.7924	13.0553	**38.7072**	**2.7853**	34.1980	236
Normal Estimation	7.7216	1.6686	6.6608	0.3319	6.2779	139
Sobel Filter	1.9074	0.2801	1.7846	0.0610	1.7190	276
Sparse Matrix Vector Multiplication	0.244	0.0427	0.205	0.0380	0.1661	148

Table 4.3 shows the result of design space exploration using the constructed Bayesian surrogate model. Once the surrogate model is built, it is possible to find its optimum point. Since it is relatively inexpensive to examine the surrogate model, it is possible to perform design exploration more efficiently by evaluating the top choices that the model suggests. For

instance, in Table 4.3, the top 5 variable choices suggested by the model are also explored. The top-5 evaluation can achieve better performance in terms of finding more optimal design choices while it increases the number of samples drawn from the system by 5 iterations which is insignificant compared to the number of samples needed to build the surrogate model.

Although BO provides acceptable results, it almost always under-performs compared to GWO in both optimum mode and top-5 evaluation mode. Moreover, it usually needs more samples drawn from the environment. This can be explained by the quality of the random samples drawn from the design space. Since GWO performs a guided search in the design space environment, it can find the outliers and anomalies when searching for the optimum point. The hybrid model described in the next section helps to overcome the poor performance of BO by leveraging the strengths of GWO.

4.5.4 Hybrid GWO-BO

Table 4.4 shows the results of design space exploration using the hybrid GWO-BO method. The hybrid model outperforms both BO and GWO in finding better results in terms of average and standard deviation of latency. Note that in this experiment, the top-5 evaluation of the surrogate model is used to generate the results.

Also note that since we are optimizing a hardware design problem, it is important to choose the algorithms that find the optimum point with fewer samples. For instance, each synthesis trial of an OpenCL design can take up to hours. Table 4.4 shows that GWO-BO method uses fewer number of samples to create the surrogate model.

Table 4.4 Design space exploration using hybrid GWO-BO for 100 trials.

Benchmark Function	Average Latency (ms)	Standard Deviation of Latency (ms)	Benchmark Latency (ms) [16]	Number of Random Samples Evaluated
BFS Dense	4.329	0.0499	4.28.4	85
BFS Sparse	12.5151	0	12.5151	85
DCT	2.5891	0.1357	2.1916	35
FIR	1.6×10^{-3}	8.6×10^{-4}	7.3×10^{-4}	140
Histogram	1.5652	0.04166	1.5481	120
Mergesort	1.9386	0.0138	1.9269	230
Matrix Multiplication	36.8913	2.5895	34.1980	130
Normal Estimation	6.4315	0.10566	6.2779	110
Sobel Filter	1.7434	0.0341	1.7190	200
Sparse Matrix Vector Multiplication	0.1671	0.0022	0.1661	100

4.6 Discussion

4.6.1 Multi-Objective Optimization and Pareto Frontier

In the previous sections of this article, the considered design space methods exploration were tuned to maximize throughput. However, hardware design space exploration is normally a multi-objective problem where designers try to jointly optimize throughput and logic utilization. In multi-objective optimization, it is not possible to find a solution that simultaneously optimizes all objective functions, so the optimization involves finding the Pareto-frontier [97]. A Pareto-frontier is defined as a set of *dominant* solutions for the optimization problem. A dominant solution happens if none of the objective functions can improve in their scores unless it is by decreasing some other objective scores. A Pareto-frontier quantifies the interaction of different objectives. This section shows that having a surrogate model facilitates *estimating* the dominant solutions.

Here, we defined the objective space as the output of objective function O mapped to normalized latency and normalized logic utilization. Also, note that the latency and logic utilization are the two objectives defined for this multi-objective optimization problem.

Revisiting Figure 4.1, the design space explorer makes a query to the synthesizer and retrieves the synthesis information. This information includes the area utilization of the system. Fitting Gaussian processes are not complicated. Thus, it is possible to create a second surrogate model for area utilization as well. Furthermore, having surrogate models for throughput and area consumption, estimating the Pareto-frontier on the surrogate models is inexpensive.

Figure 4.4 shows the comparison between the actual Pareto-frontier and estimated Pareto-frontier for the two most challenging design spaces (Matrix Multiplication and Sobel Filter). The estimated Pareto-frontier is generated using the surrogate models for throughput and area utilization. Note that the surrogate models are made using the hybrid GWO-BO method. Although estimated Pareto-frontiers are very close to the actual Pareto-frontier, they are not identical. This is because the surrogate models are estimating the behavior of the system statistically.

4.6.2 Qualitative Comparison of Design Space Exploration Methods

To study the quality of the design space exploration methods introduced previously, a fitness factor can be defined as follows:

$$F^{\mathrm{DSE}} = 1 - \left(\frac{|l_{\mathrm{avg}}^{\mathrm{DSE}} - l_{\mathrm{min}}^{\mathrm{benchmark}}|}{|l_{\mathrm{avg}}^{\mathrm{DSE}}|} \right) \tag{4.10}$$

Where F^{DSE} is the fitness of the design space exploration method, $l_{\text{avg}}^{\text{DSE}}$ is the average latency that is achieved with the method, and $l^{\text{benchmark}}$ is the latency of the benchmark. When the fitness parameter F^{DSE} is closer to 1, the quality of the design space exploration is better. In other words, the fitness factor F^{DSE} is the normalized latency that is found by a given design space exploration method.

Also, it is worth mentioning that the optimum solution (i.e. $l_{\text{min}}^{\text{benchmark}}$) is known to us when computing the fitness factor according to [16] .

Figure 4.5 compares the fitness factor of the three design space exploration methods explained in the previous sections for each benchmark function. It shows that GWO (grey bars) can find near-optimum results. Despite the good search performance of GWO, it cannot provide a surrogate model of the design space objective function. Furthermore, the quality of the results obtained with Bayesian-optimization (green bars) are not as good as those obtained with GWO in several cases (e.g. FIR, Histograms, SPMV, and Mergesort). Finally, a Hybrid GWO-BO model performs at least as well as GWO and also benefits from the surrogate modeling of the design space. Hence, these results show that the combination of GWO and BO always chooses the near-optimum settings for the benchmark functions.

4.6.3 Comparison with Previous Works

In this section, we compare our results with previous works. In [98], it is shown that a *Random Search* can be more efficient for parameter tuning than a grid search. Thus, to demonstrate the effectiveness of the methods proposed in this paper, we first compare our results with a random search strategy to show that we can find better parameter settings for an OpenCL high-level synthesis workflow. Thus, a random search method was implemented and applied to the dataset used in this paper.

Moreover, we also compared our results with the LASSO (Least Absolute Shrinkage and Selection Operator) regression model introduced in [16]. In both cases, GWO-BO finds more qualified optimum solutions faster.

Table 4.5 summarizes this comparison. The random search is done over 100 trials and the average latency as well as standard deviation of the best solution obtained in the trials are reported in the table. Furthermore, in order to make a fair comparison, the number of random samples in each trial is equal to the numbers reported in Table 4.4 for the GWO-BO algorithm. Note that GWO-BO algorithm produces better results, both in terms of average latency and its standard deviation in all benchmark functions.

Table 4.5 also includes the results obtained with the LASSO model over the whole design

Table 4.5 Comparison with previous works.

Benchmark Function	Random Search [98] Average[1] Latency (ms)	Random Search [98] Standard Deviation of Latency[1] (ms)	LASSO Model (ms) [16]	This work[2] Average Latency (ms)	This work[2] Standard Deviation of Latency (ms)
BFS Dense	4.372	0.05304	4.3825	**4.329**	**0.0499**
BFS Sparse	13.371	0.5449	13.2857	**12.5151**	**0**
DCT	2.904	1.485	2.9522	**2.5891**	**0.1357**
FIR	1.8×10^{-3}	1.03×10^{-3}	2.4×10^{-3}	$\mathbf{1.6\times10^{-3}}$	$\mathbf{8.6\times10^{-4}}$
Histogram	1.6632	0.1232	1.5715	**1.5652**	**0.04166**
Mergesort	1.9955	0.0685	2.0441	**1.9386**	**0.0138**
Matrix Multiplication	40.9993	8.8535	40.6412	**36.8913**	**2.5895**
Normal Estimation	6.5449	0.2311	6.4763	**6.4315**	**0.10566**
Sobel Filter	1.8448	0.1145	1.9331	**1.7434**	**0.0341**
Sparse Matrix Vector Multiplication	0.1729	0.0056	0.1798	**0.1671**	**0.0022**

[1] 100 trials
[2] Using hybrid GWO-BO method

space. We used the exact LASSO model of [16]. Although, all the design space information is incorporated in the LASSO model, it did not perform better than the GWO-BO model in any of the benchmark functions. We suspect that regression models such as LASSO are not suitable to model the optimum behavior, since they intrinsically smooth the outlier information.

Also, it is noteworthy that in Table 4.5, there are some extreme cases where GWO-BO offers significant advantages over the LASSO model and the random search. For instance, in BFS sparse, GWO-BO can always find the global maximum with a standard deviation of zero, while the random search and LASSO model fail to find it. Another significant example is the FIR benchmark, where the standard deviation obtained with the GWO-BO search is significantly smaller than the results obtained with random search. Likewise, in the Matrix Multiplication benchmark, the GWO-BO method can find significantly better latency than the random search or LASSO model.

4.6.4 Other Related Works

GWO has recently been used for other exploration problems. For instance, in [99], the authors used the GWO algorithm for evolving Convolutional Neural Network Long Short-Term Memory (CNN-LSTM) for time series analysis. They showed that GWO can produce significantly better results than other meta-heuristic methods. Moreover, as discussed in

Section.4.5.2, they manipulated the $\hat{\mathbf{a}}$ parameter of the GWO algorithm to reach the best trade-off between exploration and exploitation.

Similarly, in [100], the authors used the GWO algorithm to explore and find the best hyper-parameter settings for LSTM-based language models. Their experimental results show that the GWO algorithm with 15 search agents can find the global optimum of the search space. Also, in [101], the authors used a modified GWO algorithm to solve the problem of workflow scheduling in the cloud. Their results show that the modified GWO algorithm can outperform the common scheduling approaches in terms of power consumption and cost.

4.7 Conclusion

This paper describes the implementation of a design space exploration tool to optimize OpenCL kernels for FPGA targets. Three design space exploration methods (GWO, BO, GWO-BO) have been implemented. Grey Wolf Optimization (GWO) is a meta-heuristic that produces reasonable performance for design space search problems, but it does not provide a surrogate model of the system. Bayesian Optimization (BO) is another well-known method that can provide a surrogate model of the system, however, it does not perform as well as GWO. This paper ultimately proposes a novel hybrid GWO-BO method that combines the BO and GWO in a way that the BO uses the samples explored by GWO to create a surrogate model. According to the benchmark tests, the hybrid method outperforms GWO and BO. Finally, we suggest that the surrogate model produced by the hybrid GWO-BO model can also be used to estimate the Pareto-frontier for multi-objective area/throughput optimization.

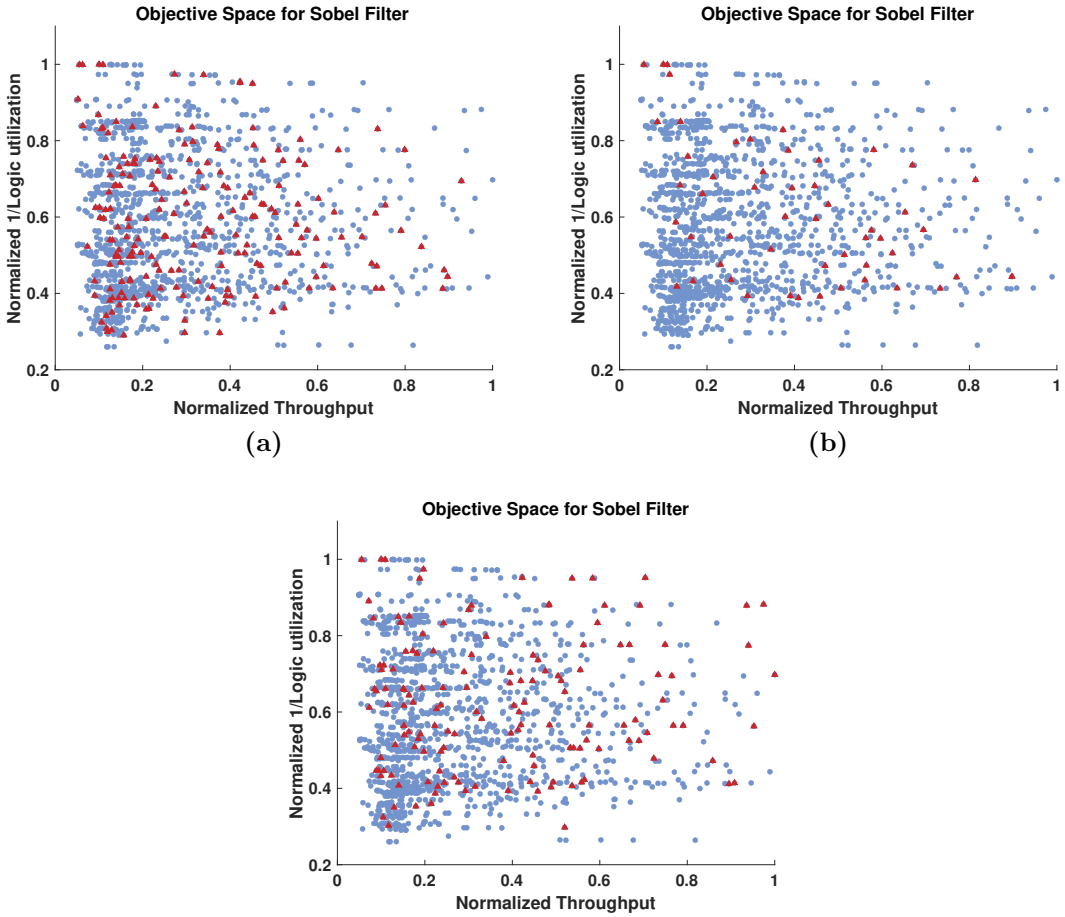


Figure 4.3: Effect of $\hat{\mathbf{a}}$ on the exploration/exploitation trade-off. Red triangles are the explored points by wolves and blue circles are the design space. (a) $\hat{\mathbf{a}}$ is linearly decreased from 2 to 1 keeping the wolves in exploration mode. (b) $\hat{\mathbf{a}}$ is linearly decreased from 1 to 0 keeping the wolves in exploitation mode. (c) $\hat{\mathbf{a}}$ is linearly decreased from 2 to 0 in order to balance the exploration and exploitation which results in finding better variables to maximize the throughput

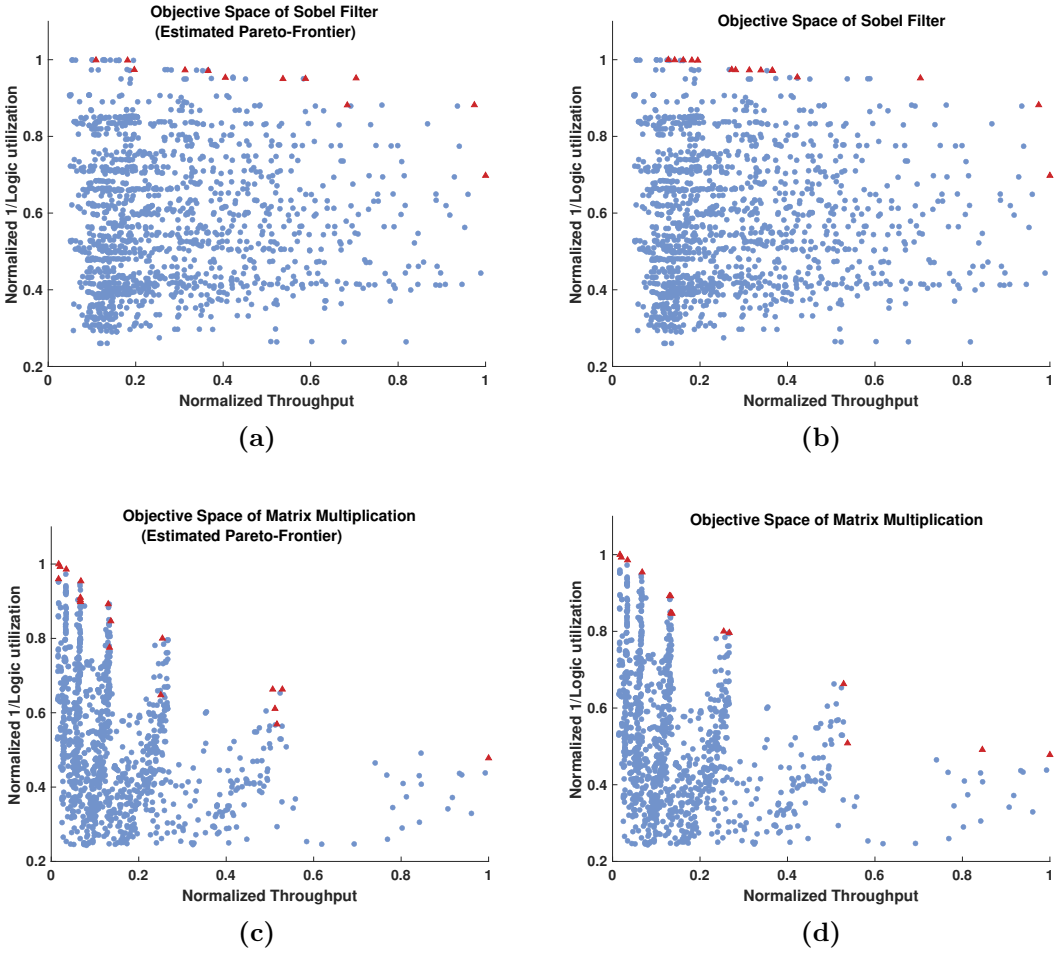


Figure 4.4: Red triangles in (a) and (c) are the Pareto-frontier estimated by surrogate models for throughput and area utilization. Red triangles in (b) and (d) are the actual Pareto-frontier of the design space according to benchmark [16]

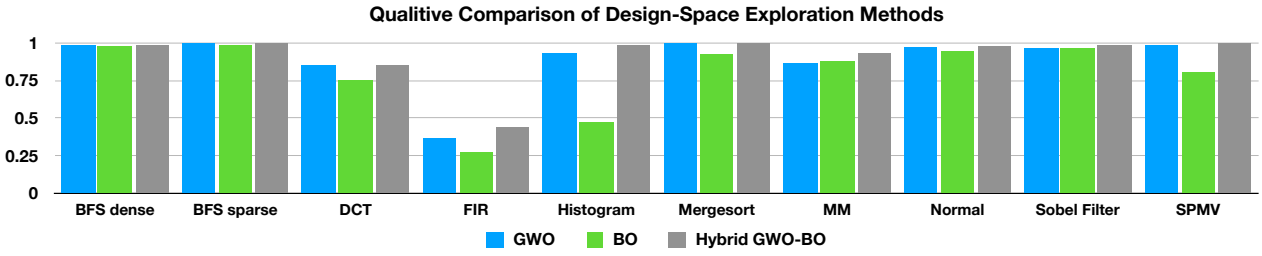


Figure 4.5: Qualitative comparison of design space exploration methods. The vertical axis shows the fitness F^{DSE}. Higher fitness factors correspond to better solutions

CHAPTER 5 ARTICLE 3 : STATISTICAL HARDWARE DESIGN WITH MULTI-MODEL ACTIVE LEARNING

This chapter is a reproduction of an article submitted to IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems on April 9, 2023. The following version of the paper was revised as per the reviewer's comments.

- Ghaffari, A., Asgharian, M. and Savaria, Y., 2023. "Statistical Hardware Design with Multi-model Active Learning". Revised version submitted to IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems.

5.1 Abstract

With the rising complexity of numerous novel applications that serve our modern society comes the strong need to design efficient computing platforms. Designing efficient hardware is, however, a complex multi-objective problem that deals with multiple parameters and their interactions. Given that there is a large number of parameters and objectives involved in hardware design, synthesizing all possible combinations is not a feasible method to find the optimal solution. One promising approach to tackle this problem is statistical modeling of a desired hardware performance. Here, we propose a model-based active learning approach to solve this problem. Our proposed method uses Bayesian models to characterize various aspects of hardware performance. We also use transfer learning and Gaussian regression bootstrapping techniques in conjunction with active learning to create more accurate models. Our proposed statistical modeling method provides hardware models that are sufficiently accurate to perform design space exploration as well as performance prediction simultaneously. We use our proposed method to perform design space exploration and performance prediction for various hardware setups, such as micro-architecture design and OpenCL kernels for FPGA targets. Our experiments show that the number of samples required to create performance models is significantly reduced while maintaining the predictive power of our proposed statistical models. For instance, in our performance prediction setting, the proposed method needs 65% fewer samples to create the model, and in the design space exploration setting, our proposed method can find the best parameter settings by exploring less than 50 samples.

5.2 Introduction

Efficient hardware design is essential for various industrial and scientific applications. With the emergence of new applications such as machine learning, the Internet of Things (IoT), and 5G wireless communication, there is a strong need to design computing platforms that provide the required computational power. Designing efficient hardware is challenging and requires considering various hardware and software parameters and their interactions. In most cases, designing efficient digital hardware is a multi-objective optimization problem where the designers try to optimize multiple competing objectives, such as energy consumption and throughput. Moreover, performing a brute-force search to find the optimum design is not feasible, because synthesizing complex hardware designs is very time-consuming. In addition, designers must test a large number of parameter combinations to find the optimum solution.

Another prominent challenge of efficient hardware design is that the objective function is not smooth, and most parameters are discrete. For example, the memory delay, the number of floating-point arithmetic units, and clock frequencies are parameters that need to be tuned for throughput. Since these parameters are discrete, the objective function is not differentiable. Thus using gradient-based methods such as Stochastic Gradient Descent (SGD) to find the optimum parameter set is not possible.

For years, meta-heuristic methods, such as Genetic Algorithm, were used to perform design space exploration [92, 102]. Although these meta-heuristic algorithms can find the optimum setup in various settings, they cannot provide a model that predicts the performance. Moreover, the heuristic methods cannot provide an abstract model to study the interactions and inter-dependencies between the parameters and their effect on the desired performance metrics. Researchers proposed Bayesian models for design space exploration tasks to solve this issue. Bayesian optimization is not only capable of finding the optimum parameter setup but is also able to reveal the interactions and inter-dependencies between the parameters of the design.

Although research on Bayesian models has produced remarkable results for design space exploration as well as abstract modeling of parameter inter-dependencies, they incorporate two fundamental shortcomings. First, there is a need for a feedback mechanism from a real-world application to update the Bayesian model to improve accuracy. Second, Bayesian models are application specific, meaning that a model trained for a particular task cannot be used for another. Thus, there is a need to propose a *transfer learning* strategy to use currently trained models to predict the performance of applications that were not previously exposed to the model. Using transfer learning is crucial in hardware design since it provides the

ability to perform design space exploration for emerging applications using prior knowledge acquired from previous experiences and applications.

Here we propose a novel methodology that solves the two shortcomings mentioned above. Besides the Bayesian modeling approach, we suggest using Bayesian active learning [21, 53] to increase our proposed statistical model accuracy. Bayesian active learning is an iterative machine learning algorithm that actively selects some informative samples to construct the Bayesian model of the objective function. The iterative nature of the Bayesian active learning method helps the Bayesian model to learn the underlying structure of the data gradually. The ultimate goal of Bayesian active learning is to provide a surrogate function that closely resembles the system's behavior with the minimum number of samples. In our proposed methodology, once a Bayesian model is created, it can be updated using a feedback mechanism that is provided by a real-world application. For instance, a Bayesian surrogate model can be updated using a processor or computing cluster performance metrics reflecting its performance for different computation loads. Furthermore, we propose using the Bayesian transfer learning [103], where the information of one domain can be transferred to another domain to improve the accuracy of the statistical model of the target task. Transfer learning is particularly useful in hardware design in scenarios where the target application is novel and there are limited information and data on the performance of the hardware in that application. In such cases, the information of previously trained hardware models can be leveraged to infer the performance of the new application.

To further reduce the number of samples needed to make hardware performance prediction, we suggest using *Gaussian regression bootstrapping*. Once we have a Bayesian model of the hardware, it is possible to re-sample the model in order to acquire new composite samples to perform other types of statistical analysis. In this paper, we used Gaussian regression bootstrapping to generate new data to perform various regression analyses of the hardware model and showed that the results based on bootstrapped samples could closely resemble the analyses done on the real data.

To summarize, this paper makes the following contributions:

- We propose a *model-based active learning* framework that generates statistical models of different aspects of hardware performance.
- We propose *Bayesian transfer learning* to effectively perform hardware design space exploration using prior hardware performance information.
- We use a *Gaussian regression bootstrapping* method to effectively estimate the behavior of the hardware from the limited samples acquired from real-world applications.

To the best of our knowledge, this is the first time that model-based active learning has been used in conjunction with Bayesian transfer learning to provide accurate statistical hardware models for design space exploration. Furthermore, this is the first time that Bayesian regression bootstrapping is used to estimate the behavior of specific hardware better.

The rest of the paper is structured as follows. Section 5.3 reviews related works in the hardware design space exploration and performance prediction. We introduce our methodology in Section 5.5. In Section 5.4, we distinguish two important settings, notably design space exploration and performance modeling. In Section 5.6, we show that our proposed methodology is effective for both design space exploration and performance modeling.

5.3 Related Works

Exploring various design parameter sets in hardware design is a ubiquitous problem. The most obvious but inefficient solution is brute-force analysis or exhaustive search. The complexity of brute-force analysis increases exponentially with the number of parameters, making this method infeasible in practical problems. For years, researchers have used meta-heuristic methods such as Genetic Algorithms (GAs) [102] to perform design space exploration and obtain the Pareto optimal solutions. Although meta-heuristic methods effectively find optimum solutions, they do not provide a mathematical model reflecting inter-dependencies between parameters and their effects on the objectives.

Another valuable design space exploration approach is Bayesian Optimization (BO) [21]. Bayesian Optimization is especially interesting for design space exploration applications since (i) it provides a Bayesian surrogate model that helps to understand the inter-dependencies of the parameters and (ii) it is an iterative method that incorporates information from previously observed samples in the surrogate model. BO has been widely used in hardware design space exploration. For instance, in [22], Mehrabi et al. found that BO have a superior performance for design space exploration of `C/C++` High-Level Synthesis (HLS) compared to traditional search methods. Likewise, Reagen et al used BO in [23] to optimize hardware accelerators for deep neural networks. They reported that the BO method outperforms traditional design space methods in all metrics of interest, such as accuracy and energy consumption. Furthermore, in [20], Lo et al. BO deployed BO to tune HLS directives to achieve minimum latency.

Gaussian processes with deep kernel learning functions are used in BOOM-Explorer [37] to perform microarchitecture-aware active learning. BOOM-explorer also uses BO to solve multi-objective problems for design space exploration and finds the Pareto frontier efficiently.

Furthermore, in [104], the authors used BO with multi-task Gaussian models as surrogate functions to choose optimal EDA parameters and efficiently estimate the Pareto frontiers.

Inspired by traditional meta-heuristic approaches, researchers have combined BO and Grey Wolf Optimization (GWO) [52] to perform design space exploration for HLS of OpenCL kernels [105]. The hybrid GWO-BO method was shown to outperform both BO and traditional search methods.

Other iterative methods such as Reinforcement Learning (RL) [28] are also effective for design space exploration applications. An RL agent tunes hardware parameters through some iterations to maximize the reward function related to metrics of interest. For example, in [29], de Prado et al. used the RL method to explore parameters that optimize the latency of deep neural networks on the ARM-Cortex-A CPUs. Likewise, in [30], a time-limited reinforcement learning [31] is used to perform design space exploration for deeply pipelined OpenCL kernels of the convolutional neural networks. In [32], an Advantage Actor Critic (A2C) [33] design space exploration is proposed to minimize area subject to a timing constraint in logic synthesis. Additionally, in [34], Kao et al. proposed an RL-based design space exploration method and used the performance predicted by their cost model to compute the reward values. Moreover, they augmented their RL algorithm with a genetic algorithm to improve the design space exploration results.

Furthermore, using other statistical methods for hardware analysis and design is commonplace in the hardware community. For instance, in [11], Lee et al. studied accurate regression modeling of micro-architecture energy consumption and throughput. Moreover, in [18], Liu et al. used Random Forest for HLS design space exploration. Similarly, in [19], Singh et al. used Random Forest regression for predicting instruction per cycle for near-memory computing applications. In [94], Beltram et al. found Markov decision process effective for the design space exploration of multi-processor platforms. In [106], Partovi Nia et al. used parametric bootstrapping to perform stochastic ordering deep learning models that run efficiently on different hardware, while the inference efficiency and training efficiency are chosen as the metrics of interest.

Our proposed method is different from the previous Bayesian methods ([20, 22, 23]) as it uses Active Learning in conjunction with Bayesian Optimization to use synthesis results or any results obtained before a full logic synthesis of an RTL model called pre-RTL estimation results in [43] for creating more accurate Bayesian models. We present results showing that our method is more effective than the hybrid GWO-BO method proposed in [105]. Furthermore, we use Bayesian Transfer Learning (TL) [103] and show that the performance of the parameter search is significantly improved by incorporating information from other

correlated hardware design tasks. Our proposed method differs from [106] since it can produce Bayesian models capable of being used for *Gaussian regression bootstrapping* [107]. Also, after performing Gaussian regression bootstrapping, it is possible to produce composite data from our Bayesian surrogate model and perform statistical regression analysis similar to ([11,94]) without requiring real-world data samples.

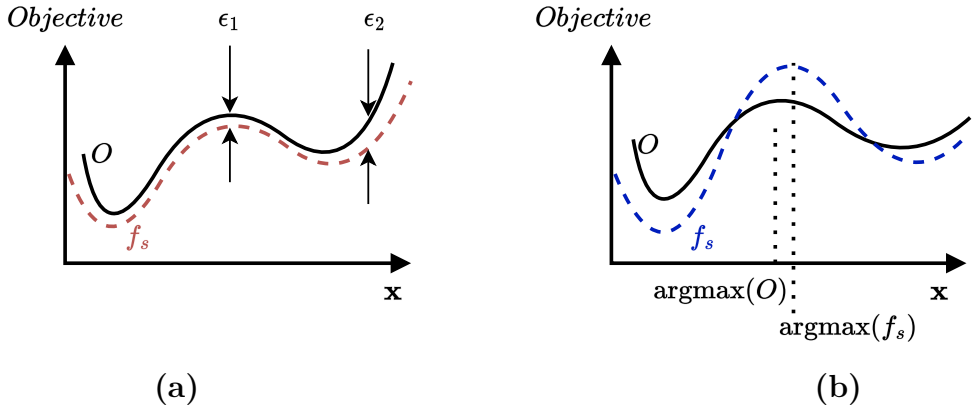


Figure 5.1: (a) Performance modeling (regression) setting where the accuracy of the model is evaluated with the Mean Squared Error (MSE) and (b) Design space exploration setting where the algorithm tries to find the parameters that correspond to the design with maximum performance

5.4 Problem Statement: Design Space Exploration vs Performance Modeling Settings

Hardware design space exploration and hardware performance modeling are two related but distinct tasks in hardware design. Our proposed statistical modeling methodology provides a unified framework for both applications that have been crucial for hardware designers historically. In the *Performance Modeling* application, the goal is to accurately *predict* the hardware performance (or value of the objective function O) for a specific hardware design parameter setting. Thus, the performance modeling application is essentially a regression problem and the accuracy of the model can be measured with *Mean Squared Error (MSE)* between the surrogate function f_s and the actual objective function O over N parameter sets $\mathbf{x}_{1:N} = \{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N\}$

$$\text{MSE} = \frac{1}{N}\sum_{i=1}^{N}\left(O(\mathbf{x}_i) - f_s(\mathbf{x}_i)\right)^2 = \frac{1}{N}\sum_{i=1}^{N}\varepsilon_i^2. \tag{5.1}$$

Figure 5.1 (a) demonstrates the performance modeling setup. The black curve is the actual objective function, while the dotted red curve is the regression model. Note that in our experiments, we report $\sqrt{\text{MSE}}/\mu_{f_s(\mathbf{x}_i)}$ to normalize MSE with respect to the mean of the surrogate function $\mu_{f_s(\mathbf{x}_i)}$.

On the other hand, in the *Design Space Exploration (DSE)* application, the goal is to find the hardware design parameter set that maximizes the objective function. Moreover, in the case of multi-objective optimization, DSE aims to find the dominant or Pareto optimal solutions. Figure 5.1 (b) shows the DSE setup. The model is accurate if

$$\arg\max(f_s) = \arg\max(O), \tag{5.2}$$

meaning that the surrogate function f_s can accurately predict the parameter set that maximizes the objective function. Also, note that here *argmax* is used for the objectives that need to be maximized, such as throughput. Likewise, the goal for the objective functions that are to be minimized, e.g. power consumption, is the same, except that arg max should be replaced by arg min.

5.5 Methodology

Statistical modeling of hardware and related applications such as design space exploration and performance estimation can be classified as Black Box Optimization (BBO) [90] problems. This is because the objective function and its behavior, such as smoothness, continuity, and convexity, are unknown. Thus, gradient-based methods such as Gradient Descent (GD) and Stochastic Gradient Descent (SGD) are not the best choice for these applications. On the other hand, derivative-free methods such as meta-heuristic methods and Bayesian Optimization are among the best choices for these applications.

Here we propose a *Model-based Active Learning* framework for statistical hardware performance modeling. We further analyze this method in the context of design space exploration and performance prediction (regression analysis). In our proposed method, we use Bayesian Optimization with Gaussian Processes. We also use Bayesian Transfer Learning and Bayesian regression bootstrapping methods to further improve such models in practical applications. Figure 5.2 shows an overview of our proposed framework. In this figure, Bayesian models are de facto Gaussian processes that are updated iteratively using active learning. These Bayesian models can be used in design space exploration or for performance prediction. The details of our mathematical modeling are elaborated on in the following sections.

We used multi-model active learning because we wanted to assign one Gaussian process to each objective. The benefits of this approach are as follows. (i) Treating each objective separately as a Gaussian process simplifies mathematical computations. (ii) All Gaussian processes contribute to choosing the next data samples in the queries performed by our active learning procedure; see Section 5.5.1. (iii) Applying the transfer learning approach is easier for each objective when estimated by a single Gaussian process; see Section 5.5.3. The details of our mathematical modeling are elaborated on in the following sections.

5.5.1 Multi-Model Active Learning

We use active learning to update the Gaussian process models. Our proposed framework uses multiple Bayesian models for various objectives. For instance, power consumption and throughput are each associated with their corresponding models. The algorithm starts from a random set of parameters for all models. It then queries the synthesis tool (or a so-called pre-RTL estimator in simple cases) and creates initial Gaussian processes. Next, the algorithm evaluates the Gaussian processes and gathers each model parameters candidate to be sent to the synthesis tool (or pre-RTL estimator) in the next query. This process continues iteratively until a particular stopping criterion is met. Furthermore, when used with active learning, the algorithm chooses the next set of parameters using the Gaussian models that are also affected by other pre-trained Gaussian processes, as explained in Section 5.5.3. Also, note that our proposed algorithm considers the direction of optimization of each objective

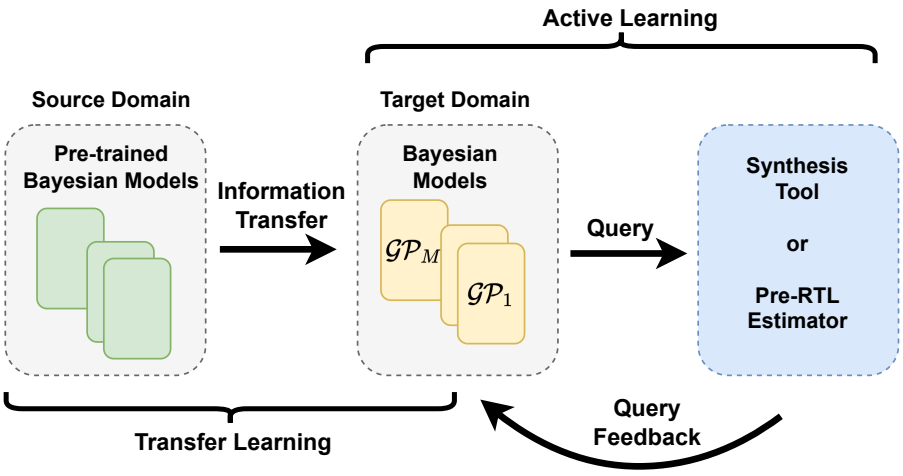


Figure 5.2: Statistical modeling of hardware using *Multi-model Bayesian Active Learning*. The framework also incorporates an optional *Bayesian Transfer Learning* setup to improve the accuracy of the Bayesian models

function. For instance, designers usually aim to minimize power consumption while maximizing the throughput. Therefore, our proposed algorithm chooses the sets of parameters that respect the optimization constraints of each objective function. Having multiple models associated with multiple objectives enables us to study the Pareto efficiency of the solutions. Section 5.6.1 reports experimental results that validate the proposed method. Algorithm 9 provides a pseudo-code for our proposed algorithm.

5.5.2 Bayesian Optimization with Gaussian Processes

Bayesian Optimization [21] is a powerful tool to find the extrema of an objective function that is hard to evaluate or does not have a closed mathematical form. Statistical modeling of hardware implementations deals with both problems simultaneously. It is well-known that synthesizing hardware models can take a very long time, and generally, there is no closed-form expression of the power consumption and latency of the resulting hardware models.

In Bayesian Optimization, the goal is to find a surrogate function f_s that closely resembles the behavior of the desired objective function O. Note that in Bayesian Optimization, f_s is a stochastic process.

Let us assume $\mathbf{x}_i$ is the i^{th} sample, and $O(\mathbf{x}_i)$ is the evaluation of the objective function for that sample. Using Bayes theorem, for a collection of N samples such as $C_{1:N} = \{\mathbf{x}_{1:N}, O(\mathbf{x}_{1:N})\}$, we have

$$P(O|C_{1:N}) \propto P(C_{1:N}|O)P(O), \tag{5.3}$$

where $P(O|C_{1:N})$ is the posterior distribution. The posterior distribution P incorporates probabilistic information about the objective function O. Also, the commonplace assumption

Algorithm 9: Multi-Model Bayesian Active Learning.

Step 1. Randomly choose the first set of parameters $\mathbf{x}_{1:N} = \{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N\}$ for query.
while ***stop criterion*** *is not satisfied* **do**
  Step 2. Get the query Objectives $O(\mathbf{x}_{1:N})$.
  Step 3. For all $\{\mathcal{GP}_1, \mathcal{GP}_2, \ldots \mathcal{GP}_M\}$, calculate new covariance matrix using eq (5.5)
  Step 4. Update all $\{\mathcal{GP}_1, \mathcal{GP}_2, \ldots \mathcal{GP}_M\}$ using eq (5.4)
  Step 5. If transfer learning is needed: Update $\{\mathcal{GP}_1, \mathcal{GP}_2, \ldots \mathcal{GP}_M\}$ mean vector and covariance matrix using eq (5.6)
  Step 6. Evaluate all $\{\mathcal{GP}_1, \mathcal{GP}_2, \ldots \mathcal{GP}_M\}$ and update the set $\mathbf{x}_{1:N}$
return $\{\mathcal{GP}_1, \mathcal{GP}_2, \ldots \mathcal{GP}_M\}$

in Bayesian optimization is that the objective function is continuous. This assumption does not hold in our experimental setup. However, our goal is to perform Bayesian optimization on a discrete objective function by fitting a continuous posterior distribution P that passes through the discrete points of the objective function O, see [24].

We note that the surrogate function f_s is a stochastic process. Let us further assume that f_s is a Gaussian Process that provides a mean and a standard deviation at each point. i.e.

$$f_s(\mathbf{x}) = \mathcal{GP}(\mu(\mathbf{x}), \mathbf{K} = k(\mathbf{x}, \mathbf{x}')). \tag{5.4}$$

This assumption implies that the sampling process of $O(\mathbf{x}_i)$ is noisy with Gaussian distribution $\mathcal{N}(0, \sigma^2)$. Also note that a time-continuous stochastic process A_t is a Gaussian process if the finite set of $(A_{t_1}, A_{t_2}, \ldots, A_{t_k})$ is a multivariate Gaussian random variable. This also implies that every linear combination of $A_{t_1}, A_{t_2}, \ldots, A_{t_k}$ is a univariate normal distribution. Assuming a normal distribution as prior distribution is common in the hardware community, as seen in [20, 22, 23].

In eq (5.4), $\mathcal{GP}$ denotes a Gaussian process with a mean function μ and a covariance operator $\mathbf{K}$. Here, we use the squared exponential covariance kernel k defined below to construct the covariance matrix $\mathbf{K}$

$$\mathbf{K}_{ij} = k(x_i, x_j) = \exp(-\frac{|x_i - x_j|^2}{2}) \tag{5.5}$$

There are other types of covariance kernels such as *Matern kernels* [108] that are suitable for non-smooth data. Although studying the effect of various covariance kernels is not the main focus of this work, we show in Appendix B that our proposed method for hardware performance modeling is robust in terms of changing the covariance kernel to Matern.

5.5.3 Transfer Learning

We propose using *Transfer Learning* to transfer prior information from a *source* surrogate function to a *target* function. Let us denote source domain $\mathcal{D}_{\text{source}}$ that contains design parameters of our hardware design problem $\mathbf{x} = \{x_1, x_2 \ldots x_p\}$. The surrogate function of source task $\mathcal{T}_{\text{source}}$ is also denoted as f_{source}. The goal is to improve the learning process of a target task $\mathcal{T}_{\text{source}}$ with the surrogate function f_{target} and domain $\mathcal{D}_{\text{target}}$. Note that our application considers *Inductive Transfer Learning*, where $\mathcal{T}_{\text{source}} \neq \mathcal{T}_{\text{target}}$ and $\mathcal{D}_{\text{source}} = \mathcal{D}_{\text{target}}$. This means that in our experimental setup, although the tasks and their surrogate functions are different, the design parameters $\mathbf{x} = \{x_1, x_2 \ldots x_p\}$ of our hardware design problems are the same.

In our proposed method, we consider *Inductive Gaussian Process Transfer Learning* where we allow the mean function and covariance operator of the source surrogate function to affect the target surrogate function. To this end, we define

$$\begin{cases} \mu^{\mathrm{TL}}(\mathbf{x}) = \mu_{\mathrm{target}}(\mathbf{x}) + \lambda_1 \mu_{\mathrm{source}}(\mathbf{x}) \\ \mathbf{K}^{\mathrm{TL}} = \mathbf{K}^{\mathrm{target}} + \lambda_2 \mathbf{K}^{\mathrm{source}} \end{cases} \tag{5.6}$$

where λ_1,λ_2 are hyper-parameters that regularize the magnitude of the source information transferred to the target.

We emphasize that we only use transfer learning for design-space exploration experiments where finding the position of the objective function optimum point is more important than its value, as explained in Section 5.4. This method is beneficial when the $\mathcal{T}_{\mathrm{source}}$ and $\mathcal{T}_{\mathrm{target}}$ are correlated. We will show in Section 5.6.1 that this assumption is true for our intended application.

5.5.4 Gaussian Regression Bootstrapping

In statistical hardware modeling problems, the aim is to predict the behavior of an objective function O. Examples of such objective functions are power consumption and latency. In practice, we have access to limited samples of this objective function because synthesizing a hardware design is a very time-consuming process. Thus, we can use the Bayesian surrogate function f_s to evaluate the behavior of the objective function O for new sets of parameters. This method is commonly known as *Gaussian Regression Bootstrapping* [107] and its theoretical aspect is studied in [109]. This is specifically beneficial because, unlike O, sampling f_s is inexpensive. Note that f_s is a Gaussian process incorporating mean and variance information. Here, we can use the mean function μ of f_s for resampling the objective function O

$$O(\mathbf{x}) = \mu(\mathbf{x}) + \varepsilon(\mathbf{x}) \tag{5.7}$$

which provides an inexpensive evaluation of O for hardware parameter setups that are not actually synthesized. Note that ϵ denotes the error of a $\mathcal{GP}$'s with a mean function equal to zero and a covariance operator equal to $\mathbf{K}$.

5.6 Results and Discussions

Here, we discuss the performance of our proposed method and compare it with previous works. We show that our proposed method not only outperforms other algorithms for design space exploration applications, but it also provides a viable performance prediction in the regression setting.

5.6.1 Design Space Exploration (DSE) Setting

We study the performance of our algorithm for DSE setting for two applications. In Section 5.6.1, we study the multi-objective optimization of OpenCL kernels on FPGA targets, and in Section 5.6.1 we show the effect of transfer learning on design space exploration of the CPU micro-architecture design.

Case Study I: Multi-Objective Optimization of OpenCL Kernels on FPGA Targets

As studied previously in [16, 105], design space exploration of the OpenCL kernels on FPGA targets is a challenging task that involves optimizing multiple competing performance objectives such as throughput and area utilization using various design parameters. These design parameters include, but are not limited to, the number of work-groups, the number of compute units, SIMD size, unrolling, etc. Furthermore, such kernels must be optimized for both area and throughput. Having two objectives, i.e., logic utilization and throughput, makes the

Table 5.1 Design space exploration using Bayesian Active Learning and Comparison with previous works.

Benchmark Function	Random Search [98] Average Latency (ms)	LASSO Model (ms) [16]	Hybrid GWO-BO [105] Average Latency (ms)	This work Latency (ms)
BFS Dense	4.372	4.3825	4.329	**4.2804**
BFS Sparse	13.371	13.2857	12.5151	**12.5151**
DCT	2.904	2.9522	2.5891	**2.1916**
FIR	1.8×10^{-3}	2.4×10^{-3}	1.6×10^{-3}	$\mathbf{7.11\times10^{-4}}$
Histogram	1.6632	1.5715	1.5652	**1.5481**
Mergesort	1.9955	2.0441	1.9386	**1.9269**
Matrix Multiplication	40.9993	40.6412	36.8913	**34.198**
Normal Estimation	6.5449	6.4763	6.4315	**6.2779**
Sobel Filter	1.8448	1.9331	1.7434	**1.719**
Sparse Matrix Vector Multiplication	0.1729	0.1798	0.1671	**0.1661**

problem a multi-objective problem where the *Pareto Frontier* must be found by the proposed algorithm. To do so, we create a Bayesian active learning multi-model for throughput and area according to Algorithm 9.

The idea behind using our proposed method is to iteratively select new design parameters and configure the application based on them in order to find the optimal set of parameters that achieve the required performance. As shown in Algorithm 9, the Bayesian models of area utilization and latency are trained on a set of initial design parameters and their corresponding performance metrics using Gaussian processes. Then, these models are used to model the performance of other design configurations and also as a guide for the selection of new parameters to be evaluated by the Bayesian model. By using active learning, the optimization process proceeds to update the Bayesian model based on the new samples evaluated by the model and it continues until satisfactory results are achieved for the performance metrics. For this experiment, we retain the five candidate solutions defined by a set of design parameters that give the best score according to the surrogate function. More generically, there could be more than one objective function, in which case each objective function would point to the five best designs. Finally, the associated *parameter sets* producing the best performance in the surrogate models are used to update the Bayesian models in each active learning round.

Figure 5.3 shows the objective space of the matrix multiplication OpenCL kernel[1]. The red triangles show the set of dominant solutions, also known as Pareto frontier [97] that is estimated by our proposed method. The estimated Pareto frontier resembles the actual Pareto frontier in [16].

Table 5.1 shows the performance of the proposed method when optimizing the OpenCL kernels for latency. Our proposed method outperforms all the previously reported methods to find the optimum parameters for achieving the best latency results in various OpenCL benchmarks. Also note that in order to do a fair comparison, we used the same number of evaluated data samples as reported in [105].

It is worth mentioning that, as discussed in Section 5.5.3 we only consider ***inductive*** transfer learning where the design parameters $\mathbf{x} = \{x_1, x_2 \dots x_p\}$ of our hardware design problems are the same. Thus, we did not use transfer learning for OpenCL kernels on FPGA since these benchmarks have different design parameters and do not satisfy the inductive condition of our proposed transfer learning approach.

[1]For a complete list of matrix multiplication HLS design parameters, refer to **Appendix A**.

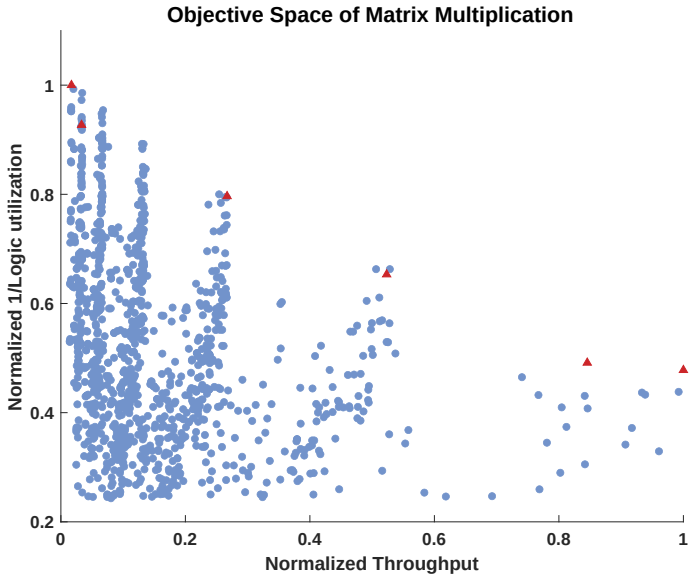


Figure 5.3: Objective space of the Matrix Multiplication OpenCL kernel is demonstrated by the blue dots. The Pareto frontier estimated by our proposed multi-model active learning method is depicted with red triangles

Case Study II: Design Space Exploration of Micro-Architectures Using Transfer Learning

Design space exploration of micro-architectures is one of the most prominent problems of digital hardware design. Similar to other design space exploration problems, the number of possible designs grows exponentially with the number of parameters (See Appendix A for the list of design parameters). As such, finding the optimal design parameters that meet the performance requirement is a challenging and time-consuming task that requires significant computational and synthesis resources. Thus, there is a need to find performance models with a minimum number of queries for the synthesis tool. In order to show that our proposed algorithm performs efficiently in this setup, similar to [11], we consider the SPEC2k [110] benchmarks to evaluate our proposed methodology. More specifically, we use the ammp, applu, equake, gcc, gzip, mesa, twolf and jbb benchmarks. The SPEC2k dataset exposes twelve parameters that can be tuned to optimize performance. These parameters are included but not limited to the cache size, the memory latency, the fixed-point and floating-point latency, etc. [11]. Moreover, considering the number of parameters, the design space for

each benchmark is about *one Billion* design points, which makes it impossible to do design space exploration using exhaustive search methods such as brute force.

Similar to Section 5.6.1, we can consider the design space exploration of the micro-architectures as a multi-objective problem and illustrate the dominant solutions using a Pareto frontier. Figure 5.4 shows a selected group of samples for objective space for SPEC2k 3-D graphics benchmark (mesa). In this figure, the Pareto frontier shows the trade-off between the Billion Instructions per Second (BIPS) rating and the normalized power, confirming that a faster microprocessor needs to consume more power. Also note that the red dots in Figure 5.4 are determined by our proposed Bayesian multi-model active learning of power consumption and instructions per second (BIPS) rating created using 600 samples. As discussed in Section 5.6.2, our proposed method considerably reduces the number of samples needed to perform regression analysis compared to the method proposed by [11].

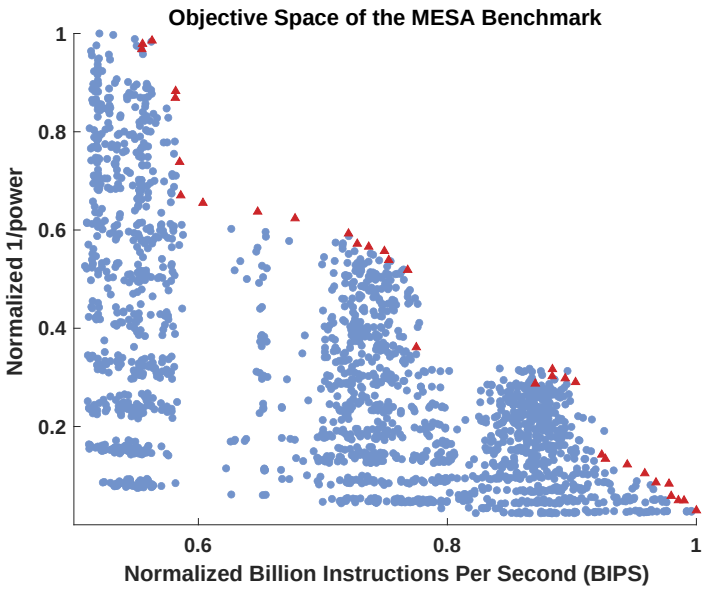


Figure 5.4: Selected samples of the objective space for the SPEC2k 3-D graphics benchmark (MESA) demonstrated by the blue dots. The estimated Pareto frontier obtained with our proposed multi-model active learning method is depicted with red triangles. The statistical model for estimating this Pareto frontier uses 600 data samples

Now we show that Bayesian Transfer Learning, as introduced in Section 5.5.3, can further reduce the number of samples required for design space exploration. We use transfer learning as a technique to leverage knowledge gained from training Bayesian models on a source task

to improve the modeling process of a target task. Let us consider the SPEC2k Java Business Benchmark (jbb) as the source task $\mathcal{T}_{source}$ and show that the information of its Gaussian surrogate function can be used to reduce the number of samples needed to perform design space exploration for other benchmark tasks. Our approach for transfer learning in this context is to use a Bayesian model that is trained using SPEC2k Java Business Benchmark (jbb) and use its Gaussian process information to affect the training of other target tasks as explained in Section 5.5.3. By doing so, the number of design evaluations is reduced, which leads to faster and more efficient design space exploration.

Note that the performance results of a micro-architecture on various benchmark tasks are strongly correlated. This means that we expect a powerful microprocessor to perform better in all tasks. This correlation is a crucial characteristic of design space exploration that can be exploited by inductive transfer learning. Table 5.2 shows this phenomenon for our selected SPEC2k benchmark tasks with the SPEC2k jbb benchmark. Note that small *pvalues* in Table 5.2 shows that the correlation results are statistically significant. In other words, it confirms the hypothesis that there is a strong correlation between the metric performance of the considered tasks.

Table 5.3 shows the experimental results for design space exploration using transfer learning for different target tasks where the source task is the SPEC2k jbb benchmark. The numbers reported in the table indicate that transfer learning can achieve, on average, 20% faster convergence to the desired BIPS. For this experiment, in eq (5.6), $\lambda_2 = 0$ and λ_1, is set to 0.5 at the beginning of the iterations and linearly decreased to zero through the course of iterations. Also note that each iteration is a query to the synthesis environment, each of which consists of five samples, to create the Gaussian processes. We further emphasize that the results reported in Table 5.3 is the effect of transfer learning on the design-space exploration problems and the transfer learning approach has a limited effect on performance prediction problems.

5.6.2 Gaussian Regression Bootstrapping and Performance Prediction Models

Bootstrap is a resampling technique in statistics that allows one to create simulated data from a given set of samples. In our proposed method, when the Bayesian models are created using Gaussian processes, it is possible to use such probabilistic methods to create simulated data that closely resemble the behavior of the hardware. Gaussian regression bootstrapping can be used in various contexts such as bioinformatics [107].

The Gaussian regression bootstrapping involves fitting Gaussian process regression to a set of performance data and using this model to resample simulated data to create new statistical

Table 5.2 Correlation of the SPEC2k benchmarks with SPEC2k Java Business Benchmark (jbb).

Benchmark Task (BIPS)	Correlation (ρ)	pvalue
ammp	0.6395	3.25×10^{-230}
applu	0.7523	0
equake	0.9100	0
gcc	0.9397	0
gzip	0.8469	0
mesa	0.8720	0
twolf	0.8920	0

Table 5.3 Design space exploration of the SPEC2k benchmarks for the Billions of Instructions Per Second (BIPS) performance.

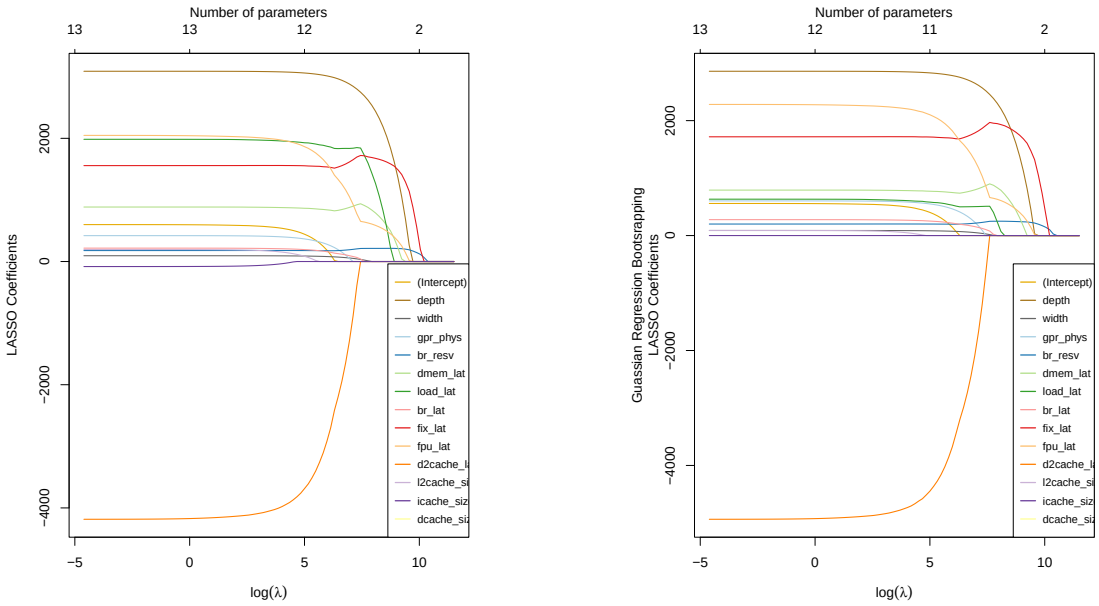
Benchmark Task	BIPS	Number of Samples without Transfer Learning	Number of Samples with Transfer Learning
ammp	1.302	35	25
applu	1.060	115	100
equake	1.027	15	15
gcc	1.009	20	15
gzip	1.005	25	15
mesa	2.012	25	20
twolf	1.138	20	15

Table 5.4 Comparison of the Predictive power of regression models with or without using Gaussian regression bootstrapping on the SPEC2k AMMP benchmark.

	Original Regression Setting $\sqrt{\text{MSE}}/\mu$	Regression using Gaussian Bootstrapping Simulated Data $\sqrt{\text{MSE}}/\mu$
Linear Regression Instructions Per Second	0.056	0.061
Linear Regression Power Consumption (mW)	0.45	0.46
Random Forest Instructions Per Second	0.017	0.055
Random Forest Power Consumption (mW)	0.050	0.14

models. This allows quantifying the uncertainty in the data and improves the accuracy of the statistical modeling for performance prediction models. The simulated data that is generated by Gaussian regression bootstrapping can be used in a variety of statistical analyses to predict the behavior of the hardware. For instance, we show that Gaussian regression bootstrapping can generate robust simulated data that can be efficiently used in various regression analyses such as linear regression, LASSO, and Random Forest regression.

Here we explore the effectiveness of our proposed active learning multi-model to predict the performance of hardware benchmarks. As mentioned previously in Section 5.4, the re-



(a) **(b)**

Figure 5.5: (a) Original LASSO model with a shrinkage factor λ for the coefficients of the power consumption model produced using 4000 samples (b) LASSO shrinkage factor λ effect on the coefficients of the power consumption model that is created using 700 samples and Gaussian regression bootstrapping. The comparison shows the extent to which Gaussian regression bootstrapping is efficient in producing simulated data samples to create statistical hardware performance prediction models

gression setting is substantially different from design space exploration and requires more samples from the actual hardware to provide acceptable models. For instance, in the case of SPEC2k benchmarks, the authors of [11] show that 2000 samples are enough to create performance regression models. However, using our proposed methodology, it is possible to reduce this number to 700 samples using Gaussian regression bootstrapping which is 65% sample reduction.

Linear Regression

Linear regression is the most basic tool for the statistical modeling of hardware. The underlying assumption, of course, is that the performance of the hardware under study is a linear function of the design parameters. Linear regression can be used to develop performance models of hardware based on tuning various design parameters such as cache size, floating

point delay, etc. Table 5.4 shows the comparison of $\sqrt{\text{MSE}}/\mu$ for linear regression using 2000 samples as suggested in [11] and 700 samples using our proposed Gaussian regression bootstrapping method. We emphasize that $(\sqrt{\text{MSE}}/\mu)$ is used here to have a fair, normalized, and dimensionless metric to compare the regression results.

Random Forest

Random forest [111] is a tree-based regression method that has been used previously for design space exploration of hardware [18]. Given the non-linear nature of hardware design, non-linear regression methods, such as Random Forest, might behave better in such scenarios. In random forest regression, multiple decision trees are created, and each tree chooses a random subset of design parameters from the training data to fit a statistical model. The main advantage of Random Forest regression over linear regression is that it can model the complex and nonlinear relation between the parameters and the desired performance objective. Table 5.4 reports that the performance of the random forest regression based on simulated data using the Gaussian regression bootstrapping is close to the original Random Forest regression. This means that the random forest regression on the simulated data and actual data produces similar results.

Note that the $\sqrt{\text{MSE}}/\mu$ values in Table 5.4 are reported for the case when the covariance kernel function is squared exponential. We show in **Appendix B** that using other types of covariance kernel (such as Matern covariance kernels) for this specific application has a negligible effect on the reported results.

LASSO

The Least Absolute Shrinkage and Selection Operator (LASSO) [112] is a regularization and variable selection method that is used in regression analysis. LASSO is also used in [16] for performance modeling of hardware. Furthermore, LASSO can be used in hardware performance prediction models to identify the most important design parameters by leveraging the variable selection property of the LASSO. The variable selection of the LASSO is particularly important for hardware performance prediction models when working with large datasets with many design parameters since the inclusion of irrelevant parameters in the performance model can lead to over-fitting the hardware performance model. Figure 5.5.a shows that as the LASSO shrinkage factor λ increases, the design parameter coefficients shrink to zero. The first coefficient that collapses to zero identifies the least significant parameter in the design space and also the last parameter that collapses to zero is the most important parameter in the design space. The top axis in this figure shows the number of variables that

are not zero at the given value of λ. Figure 5.5.b depicts the behavior of the design parameter coefficients for the regression model created from simulated data using Gaussian regression bootstrapping. Comparing Figure 5.5.a and Figure 5.5.b reveals that the behavior of the Bayesian model created using simulated data and the original LASSO model are similar for different values of the LASSO shrinkage factor, λ. For instance, in both LASSO models, *'Fixed point latency'* and *'Number of reservation stations'* are the last design parameter coefficients that are shrunk to zero. Also, note a complete list of design parameters is reported in **Appendix A**. This shows that Gaussian regression bootstrapping is efficient in producing simulated data samples to create a reliable statistical model for hardware performance prediction.

5.6.3 Estimating Pareto Frontier with Multi Model Active Learning

We elaborated in the previous sections on how various performance models can be created using Bayesian multi-model active learning. Then, when Bayesian models are trained, Gaussian process bootstrapping can be used for performance modeling. Having multiple objectives, such as throughput, power consumption, and area utilization, urges us to see the hardware design problem through the aperture of multi-objective optimization and, thus, search for a set of dominant solutions in the design space. Our proposed method simplifies estimating the Pareto frontier in the multi-objective optimization of the hardware. Fig. 5.3 and Fig. 5.4 are examples of the estimated Pareto frontier for OpenCL matrix multiplication and SPEC2k MESA benchmarks, respectively.

A practical question arises here on how to know if the estimated Pareto frontier is accurate enough and includes solutions that are close to the actual Pareto frontier. There are various methods, such as General Distance and Average Distance to Reference Set (ADRS), to compare the Pareto frontiers in the literature [35, 36]. Moreover, ADRS is commonly used in the hardware community (see [37, 38]) and is defined as

$$\mathrm{ADRS}(P_s, R) = \frac{1}{|P_s|} \sum_{p \in P_s} \min_{r \in R} d(p, r), \tag{5.8}$$

where P_s is the estimated Pareto set, R is the reference Pareto set, d is Euclidean distance and $|P_s|$ denotes the cardinality of P_s. The ADRS score is 0 if $R \subseteq P_s$ and it becomes a larger positive number when elements of P_s have some distance from R. A normalized ADRS (e.g. $\mathrm{ADRS}/\mathrm{ADRS}_{\max}$) is usually reported in the literature such as [37] for comparing the accuracy of Pareto frontiers.

Table 5 shows normalized ADRS for estimated Pareto frontiers that are obtained from multi-

objective optimization of BIPS and power consumption of SPEC2K MESA benchmark. As the number of samples increases, the Bayesian models can capture the Pareto frontier more accurately. For instance, the normalized ADRS is only 5% for 800 samples.

Table 5 Normalized ADRS for Pareto frontier estimation of multi-objective optimization of BISP and 1/Power for SPEC2k MESA benchmark.

# of samples	**30**	**150**	**300**	**600**	**800**
Normalized ADRS	0.76	0.30	0.22	**0.12**	**0.05**

5.7 Conclusion

We proposed a multi-model Bayesian active learning framework to statistically model digital hardware performance. Our proposed model uses Gaussian processes to characterize the performance of hardware designs. We also used active learning to iteratively guide the sampling process in order to reduce the number of samples needed to create the statistical model. Furthermore, we used Bayesian transfer learning in order to incorporate the prior design knowledge acquired in a source application to better model a target application with fewer data samples in the design space exploration task. We further proposed using the Gaussian regression bootstrapping method to reduce the number of samples required for the performance prediction task. To show the effectiveness of our proposed method, we performed design space exploration and performance prediction for various hardware setups, such as micro-architecture design and OpenCL kernels. For OpenCL kernels on FPGA targets, our approach was able to identify the minimum latency in all benchmarks as well as predict the correct Pareto frontier. Our experiments show that our approach can detect the optimal design parameters for processors' micro-architectures with less than 50 samples in most cases. Our experiments also show that the number of samples required to create performance models significantly reduces, by 65%, while maintaining the model's predictive power. We have also performed commonplace regression analysis, such as linear regression, LASSO, and random forest regressions, using simulated data generated by Gaussian regression bootstrapping and demonstrated that our proposed statistical hardware models closely follow the actual hardware behavior in various scenarios.

5.8 Appendix A: Hardware Design Parameters

The HLS design parameters of OpenCL matrix multiplication kernel are listed in Table 5.6. Additionally, the design parameters of SPEC2k benchmark are reported in Table 5.7.

Table 5.6 OpenCL Matrix multiplication design parameters.

#	Parameter
1	number of blocks
2	Sub-dimension x
3	Sub-dimension y
4	Manual SIMD x
5	Manual SIMD y
6	SIMD
7	Number of compute units
8	Enable Unroll
9	Unroll factor

Table 5.7 SPEC2k design parameters.

#	Parameter
1	FO4 depth
2	Load/store queue width
3	Number of physical registers
4	Number of reservation stations
5	i-L1 cache
6	d-L1 cache
7	L2 cache
8	Control latency
9	Floating point latency
10	Fixed point latency
11	Load/store latency
12	Memory latency

5.9 Appendix B: Bayesian Optimization Using Matern Covariance Kernel

In Bayesian optimization, choosing a proper covariance kernel is essential for accuracy of the algorithm. Matern kernels are a class of stationary covariance function that depends on two hyper-parameters known as the length scale l and the smoothness ν. By choosing a suitable value for ν the Bayesian model can capture the smoothness and roughness of the objective function. The Matern class of covariance kernels is defined as

$$\mathbf{K}_{ij} = k_{\text{Matern}}(x_i, x_j) = \frac{1}{2^{\nu-1}\Gamma(\nu)} \left(\frac{\sqrt{2\nu}|x_i - x_j|}{l} \right)^{\nu} H_v \left(\frac{\sqrt{2\nu}|x_i - x_j|}{l} \right), \tag{5.9}$$

where Γ is the Gamma function and H_v is modified Bessel function of the second kind of order ν. A common choice for ν are $\frac{3}{2}, \frac{5}{2}$ and also when $\nu \to \infty$ the Matern kernels becomes the squared exponential kernel [108].

We repeated the experiments in Table 5.4 using Matern covariance kernels with $\nu \in \left\{\frac{3}{2}, \frac{5}{2}\right\}$ and reported the results in Table 5.8. The reported $\sqrt{\text{MSE}}/\mu$ in Table 5.8 shows that

our method exhibits a robust behavior in terms of choosing various covariance kernels for modeling SPEC2k ammp benchmark. Note that the choice of covariance kernel depends on the nature of the data and can vary in different applications.

Table 5.8 Comparison of Matern kernels vs squared exponential kernel on Gaussian regression bootstrapping on the SPEC2k AMMP benchmark.

Covariance Kernel		**Squared Exponential**	**Matern** ($\nu = 3/2$)	**Matern** ($\nu = 5/2$)
	Original $\sqrt{\textbf{MSE}}/\mu$	**Gaussian Bootstrapping** $\sqrt{\textbf{MSE}}/\mu$	**Gaussian Bootstrapping** $\sqrt{\textbf{MSE}}/\mu$	**Gaussian Bootstrapping** $\sqrt{\textbf{MSE}}/\mu$
Linear Regression Instructions Per Second	0.056	0.061	0.061	0.062
Linear Regression Power Consumption (mW)	0.45	0.46	0.46	0.46
Random Forest Instructions Per Second	0.017	0.055	0.052	0.056
Random Forest Power Consumption (mW)	0.050	0.14	0.13	0.13

CHAPTER 6 GENERAL DISCUSSION

This thesis has addressed the intricacies of hardware design, optimization, and design space exploration through the lens of artificial intelligence and statistical modeling. The work presented in the three articles contributes to a deeper understanding of how intelligent techniques can be used to optimize the hardware design process for modern applications. In the previous chapters, various machine learning and statistical approaches are proposed to model the hardware behavior and its performance. In this chapter, we try to emphasize their advantages and highlight some of their disadvantages.

6.1 Interpretation of Results

The results and findings presented in this thesis have evolved through three distinct contributions. Each contribution has shed light on different aspects of hardware design optimization.

In the first contribution, the utilization of Reinforcement Learning (RL) methods displayed their efficacy in tackling small design spaces. However, a significant limitation emerged as the training cost of the RL agent escalated significantly for larger design spaces, rendering this approach less efficient compared to other methods.

The introduction of the Hybrid GWO-BO approach in the second contribution was a notable step forward, effectively exploring design spaces and often identifying the vicinity of optimal hardware parameter sets. Nevertheless, this approach introduced an element of randomness stemming from the inherent nature of the GWO method.

The pronounced achievement among the proposed methods was the development of the multi-model Bayesian optimization integrated with active learning. This technique consistently yielded deterministic results, successfully pinpointing the optimal design parameters for OpenCL kernels for FPGA targets. Moreover, this method was particularly able to achieve Average Distance to Reference Set (ADRS) of less than 5 percent for identifying the Pareto-optimal solutions in processor micro-architecture design. This achievement is unprecedented in hardware literature, where the typical threshold for ADRS is approximately 10 percent. This outcome shows the substantial advancement in accuracy and efficiency brought forth by the proposed methodology in Chapter 5.

6.2 Addressing Research Objectives

To the best of our knowledge, this thesis provides valuable insights and significant contributions toward optimizing the hardware design procedure using statistical and machine learning techniques. The subsequent discussion presents a comprehensive evaluation of how the established research objectives have been successfully met, substantiated by the findings from the three contributions.

6.2.1 Objective 1: Automatic Implementation of Deep Learning Algorithms on Hardware and their Design Space Exploration

The first objective aimed to seamlessly integrate deep learning algorithms, especially Convolutional Neural Networks (CNNs), into hardware platforms. The published article presented in Chapter 3 validates the accomplishment of this objective. Through the development of the CNN2Gate framework, deep learning algorithms can be effortlessly implemented on FPGA targets using high-level deep learning frameworks like Pytorch. The framework's ability to extract critical information from these frameworks and perform design space exploration has been validated. Furthermore, an important discovery was made within the pursuit of this objective; the exploration of large design spaces using Reinforcement Learning (RL) approaches is unsuitable due to the RL agent's prohibitively costly training procedure. This finding underscores the necessity of seeking alternative methods to navigate expansive design spaces effectively.

6.2.2 Objective 2: Investigation of Design Space Exploration Methods with Enhanced Sample Quality for Bayesian Optimization

The second objective delved into enhancing the efficiency and effectiveness of design space exploration. The article presented in Chapter 4 dedicated to exploring design space optimization methods established the effectiveness of meta-heuristic search algorithms, particularly the Grey Wolf Optimization (GWO) method, in conjunction with Bayesian Optimization (BO). The proposed hybrid GWO-BO method addressed the challenges posed by proper sample selection, leading to improved performance. This achievement confirms the successful realization of this objective, as it provided a method to efficiently navigate complex design spaces using innovative optimization techniques.

6.2.3 Objective 3: Development of a Comprehensive Statistical Surrogate Modeling Methodology for Design Space Exploration and Performance Prediction

The third objective aimed to develop a comprehensive statistical surrogate modeling methodology that encompassed design space exploration and performance prediction. The article presented in Chapter 5 encapsulates the culmination of this objective. Through a meticulous amalgamation of various statistical and machine-learning approaches, including Active Learning, Bayesian Optimization, Gaussian Regression Bootstrapping, and Transfer Learning, a framework was established that simultaneously addressed performance modeling and prediction while facilitating efficient design space exploration. The proposed methodology demonstrated its capability to significantly reduce the number of samples required for creating predictive models, demonstrating the achievement of this objective.

To summarize, the overarching research goal of optimizing hardware design using statistical techniques has been rigorously pursued through the accomplishment of the specified objectives. The insights obtained from the three main contributions of this thesis show the successful integration of machine learning and statistical methodologies into hardware design processes, resulting in enhanced efficiency, accuracy, and adaptability. These three contributions collectively demonstrate the realization of the research objectives, substantiating their alignment with the broader research goal and marking contributions towards advancing the hardware design optimization process.

6.3 Implications and Significance of the Proposed Methods

The proposed methods in this research bear important implications in the field of hardware design optimization. These methodologies have the potential to reshape the landscape of computational hardware development by addressing critical challenges and providing novel avenues for achieving efficient and effective designs. The implications and significance of the proposed methods can be understood from various perspectives as follows.

6.3.1 Advancements in Design Efficiency

The integration of Convolutional Neural Networks (CNNs) onto hardware platforms through CNN2Gate underscores a paradigm shift in deep learning hardware acceleration. This achievement is not only crucial for improved performance but also accelerates the adoption of CNNs in real-world applications. The potential reduction in latency demonstrated in the experimental results contributes to the realization of efficient and responsive hardware systems.

6.3.2 Enhanced Exploration Techniques

The application of heuristic methods like Grey Wolf Optimization (GWO) and the integration of Bayesian Optimization (BO) signify advancements in the field of design space exploration. These approaches offer designers efficient and effective mechanisms to navigate complex design spaces. The hybrid GWO-BO approach brings forth a method that combines the benefits of both techniques, achieving a new level of performance. The elimination of randomness through active learning reinforces the reliability and stability of the exploration process, ensuring consistent and accurate results.

6.3.3 Statistical Modeling for Digital Hardware Design

The incorporation of statistical modeling, active learning, and transfer learning into hardware design optimization is a significant progress over previously reported methods. Our proposed methodology presents a holistic approach that combines various statistical techniques. This approach enables more informed design decisions through accurate performance modeling and prediction. The exploration of inductive transfer learning further improves the adaptability and robustness of statistical models across different hardware setups.

6.3.4 Cross-Disciplinary Impact

The significance of the proposed methods extends beyond hardware design into the realm of statistical and machine learning. The integration of these disciplines with hardware engineering fosters cross-disciplinary collaboration and opens new opportunities for researchers and engineers.

6.3.5 Overal Impact

The implications and significance of the proposed methods are profound and multifaceted. They span from immediate practical applications to broader advancements in design methodologies, offering valuable tools to hardware designers, researchers, and industries. The incorporation of these methods in the realm of hardware design optimization serves as a stepping stone towards the development of more efficient, adaptable, and cutting-edge computational hardware systems.

6.4 Limitations of the Proposed Methods

The research journey undertaken to optimize hardware design using various methodologies has revealed several limitations in the proposed approaches. These limitations reveal a range of challenges encountered during the exploration of different techniques.

Initially, in the first contribution, the employment of Reinforcement Learning (RL) methods for design space exploration encountered a significant limitation. Despite their potential, RL methods exhibited costly training procedures for RL agents, rendering them unsuitable for large design spaces. This constraint hindered the scalability of RL-based approaches to tackle more complex design problems.

Transitioning to heuristic methods, such as Grey Wolf Optimization (GWO), introduced its own set of limitations. Although GWO demonstrated promising results, it could not provide a closed-form mathematical model. Moreover, the inherent randomness that was present in the outcomes of GWO was adding an element of uncertainty to the results obtained.

The adoption of Bayesian Optimization (BO) aimed to address these challenges by offering a closed-form mathematical model. However, a limitation emerged in the form of reduced accuracy compared to GWO. The accuracy shortfall was a key consideration when evaluating BO's suitability for hardware design optimization.

A hybrid solution, the GWO-BO approach, was proposed in the second contribution to leverage the strengths of both methods. While this approach provided a closed-form mathematical model and improved performance over individual GWO and BO techniques, the inherent randomness issue persisted in the results obtained. This element of randomness introduced an element of variability that needed to be addressed.

To mitigate the randomness inherent in previous methods, active learning was integrated with multi-model Bayesian optimization in the third contribution. This integration succeeded in removing the randomness factor, enhancing the reliability of results. Transfer Learning was also explored as a potential solution, yet limitations surfaced due to the challenge of out-of-domain generalization. The use of inductive transfer learning posed constraints in achieving optimal performance across different domains.

Furthermore, a limitation arose in the assumption of a Gaussian distribution prior for Bayesian optimization. While this assumption simplifies the modeling process, it might not be universally accurate. The exploration of alternative methods, such as q-Gaussian distributions [113] , could potentially address this limitation by offering a more flexible and robust modeling framework.

In summary, the limitations encountered during the research journey highlight the complexity and multifaceted nature of hardware design optimization. These challenges motivate the need for continued exploration and innovation in methods to overcome these limitations and further enhance the effectiveness of hardware optimization strategies.

6.5 A Detailed Discussion on the Proposed Hardware Optimization Methods

6.5.1 Gradient Based Methods

Although gradient-based methods, such as Stochastic Gradient Descent (SGD), present advantages such as finding global maxima in various deep learning applications, their application to the context of hardware design is accompanied by important challenges. One major obstacle arises from the non-convex nature of many hardware optimization landscapes, which can lead to convergence at local minima and hinder the identification of global optima. Another important challenge is *discrete* hardware design choices that further complicate optimization. Since gradient-based methods inherently assume continuous variables, making them less suitable for discrete hardware design spaces and potentially leading to suboptimal solutions.

Moreover, accurately calculating gradients for intricate hardware models can demand substantial effort, particularly when dealing with noisy real-world data and complex variable inter-dependencies. These challenges are compounded by the sensitivity of gradient-based methods to initialization and the risk of converging to local optima, both of which are particularly relevant in hardware optimization where non-convex landscapes and multiple extrema are common. While gradient-based methods seem promising, their successful application in hardware optimization necessitates careful consideration, potential modifications, and complementary strategies to address the specific challenges posed by complex, non-convex optimization landscapes and discrete design choices. Thus, in this thesis, we have not used gradient-based methods for our hardware optimization problems.

6.5.2 Hill climbing

Hill climbing is a simple optimization technique that involves iteratively exploring neighboring solutions and moving toward the direction of improvement. While it offers certain advantages for hardware optimization, it also has notable limitations.

Hill climbing is straightforward to implement and computationally efficient, making it suitable for the initial exploration of hardware design spaces. It is particularly useful when there is a clear and relatively smooth optimization landscape, where local improvements lead to

better solutions. This method is well-suited for fine-tuning existing designs, as it can quickly converge to a locally optimal solution by iteratively refining design parameters. Moreover, hill climbing requires minimal computational resources and is amenable to parallelization, enabling the exploration of multiple solution trajectories simultaneously.

However, hill climbing has significant disadvantages that can limit its effectiveness in hardware optimization. One important drawback is its susceptibility to getting trapped in local optima. Since hill climbing only considers immediate improvements, it may miss globally optimal solutions that require making non-improving moves in the short term to achieve long-term gains. Additionally, hill climbing's reliance on local search can lead to premature convergence, preventing the exploration of distant and potentially better regions of the design space. The method is highly sensitive to the initial starting point, and different initializations can result in vastly different outcomes. Furthermore, hill climbing lacks the ability to escape from plateau regions, where the objective function remains unchanged despite small variations in design parameters. This makes it ill-suited for non-convex landscapes with flat regions, discontinuities, or complex interactions between variables.

Hill climbing with its limitations in handling complex, non-convex landscapes, susceptibility to local optima, and inability to explore distant regions of the solution space make it less effective for hardware design challenges. Considering these limitations, in this thesis, we did not use the hill climbing method as the adopted solution for design space exploration. However, in Chapter 4, we used the hill climbing method to show that the design spaces we are dealing with are non-convex.

6.5.3 Reinforcement Learning

Reinforcement Learning (RL) has attracted attention for its potential in hardware optimization, primarily due to its adaptability for various optimization landscapes. RL's capacity to learn optimal strategies by interacting with environments is particularly attractive for hardware scenarios characterized by uncertainty or dynamics. It demonstrates versatility in accommodating diverse hardware architectures and objectives, especially within high-dimensional and continuous design spaces. Furthermore, RL's ability to explore unconventional solution paths and capture complex dependencies is advantageous for optimizing various hardware aspects.

However, RL's application to hardware optimization comes with notable challenges that must be carefully navigated. Foremost, the *extensive training cost* required for RL agents demands substantial computational resources and time, often making its implementation *infeasible* for real-world hardware. Tuning RL hyperparameters and design choices adds

complexity to the process, potentially affecting the quality of learned policies. Additionally, RL faces exploration-exploitation trade-offs, particularly in scenarios with large or sparse solution spaces, which can negatively affect the discovery of optimal hardware designs. Transferring RL policies from simulation to real-world hardware raises significant issues due to the discrepancies between the two environments, challenging the generalizability of learned solutions.

In this thesis, we used RL in Chapter 3 for the design space exploration of CNNs on FPGA targets. Due to the extensive training cost of RL agents, we move forward to heuristic methods and also Bayesian optimization to allow tackling larger design space exploration problems.

6.5.4 Heuristic Methods

Heuristic methods, such as Grey Wolf Optimization (GWO), show great potential for hardware optimization, offering practicality and exploration capabilities. GWO, inspired by grey wolf social behavior, presents a straightforward approach for optimization. It requires minimal parameter tuning, making it accessible to a wide range of users. With its population-based nature, GWO can efficiently explore diverse regions of the design space, enhancing the chance of discovering optimum solutions. Heuristic methods like GWO are well-suited for hardware optimization problems characterized by complex, non-convex landscapes. They strike a balance between optimization quality and computational efficiency, making them valuable in resource-constrained scenarios.

However, heuristic methods exhibit important limitations that must be considered. These methods lack theoretical guarantees of convergence to the global optimum, relying on exploration strategies that might not guarantee finding the best solution in every instance. The performance of GWO, for instance, can be sensitive to algorithmic parameters, necessitating careful tuning for optimal results. Moreover, the random nature inherent in heuristic methods introduces variability, implying that each run may converge to different local optima. This randomness poses a challenge in assessing the reliability and consistency of results.

Another important shortcoming of the heuristic methods is that they do not provide a closed-form mathematical model of the hardware. This impedes using such optimization methods in more complicated design optimization scenarios such as transfer learning and also limits the usability of the method to discover inter-dependencies between design variables. We used GWO in Chapter 4 and also combined it with BO to mitigate its limitations.

6.5.5 Bayesian Optimization

Bayesian optimization is a promising technique for hardware optimization, offering a range of benefits with limited disadvantages. The method excels in scenarios characterized by complex, non-convex, and uncertain optimization landscapes, effectively balancing exploration and exploitation to find optimal hardware designs. By constructing a probabilistic model of the objective function, Bayesian optimization intelligently selects evaluation points, minimizing the number of costly iterations required for convergence. Its versatility in handling diverse design variables, adaptability to noisy objective evaluations, and provision of uncertainty quantification align well with the complexities of hardware systems. However, Bayesian optimization demands careful consideration of computational costs, surrogate model selection, and appropriate initial exploration to ensure its effectiveness in achieving hardware optimization objectives.

Nonetheless, Bayesian optimization is not without its challenges. Modeling errors inherent in probabilistic approaches such as BO pose a risk of converging to suboptimal solutions, and the method's efficacy is influenced by the noise-signal ratio. Ensuring suitable initial evaluations and addressing these limitations is critical for leveraging Bayesian optimization's potential to enhance hardware optimization outcomes. In this thesis, we used BO in Chapter 4 for design space exploration on the OpenCL kernels on FPGA targets. Our experiments show that BO produces inferior results compared to heuristic methods such as GWO. Thus, we suggested a hybrid GWO-BO method to benefit from the advantages of both methods.

6.5.6 Hybrid Approaches

Hybrid methods, where different optimization approaches are combined, offer an exciting way to tackle hardware design challenges. These combinations often bring together the best of both worlds, aiming to achieve more efficient and robust solutions.

Combining the strengths of Bayesian optimization (BO) and Grey Wolf Optimization (GWO) in hybrid methods can improve hardware design optimization problems. BO is good at exploring complicated design spaces and dealing with uncertainty, while GWO uses a smart search strategy. Together, they aim to find better hardware designs faster. In our proposed method, BO uses the explored samples of GWO search agents to train. The biased sampling strategy that is imposed by GWO helps BO to model optimum points.

Nonetheless, the combination of BO and GWO introduces challenges that require consideration. Ensuring seamless integration between BO's probabilistic framework and GWO's heuristic nature necessitates addressing GWO's inherent randomness and also controlling

their exploration-exploitation dynamics. These hybrid approaches demand accurate parameter tuning to achieve optimal interaction, adding an additional layer of complexity to the optimization process. In this thesis, we used hybrid GWO-BO in Chapter 4 for design space exploration of OpenCL kernels on FPGA targets and showed that the hybrid method outperforms both GWO and BO.

6.5.7 Active Learning

Active learning is a promising technique for hardware optimization that offers valuable benefits alongside certain challenges. Active learning is particularly advantageous when data collection is resource-intensive. By intelligently selecting informative samples, active learning optimizes the use of limited data points, a crucial aspect in hardware optimization where full synthesis evaluations can be costly. This approach guides the optimization process towards regions of the design space likely to find optimized solutions, contributing to more accurate predictive models and design improvement. However, active learning's success relies on the careful selection of sampling strategies and criteria, along with the consideration of initial sample quality and computational demands for dynamic sample management.

Despite its advantages, active learning comes with challenges. Its effectiveness relies on the interplay between informed sample selection and exploration strategies. Poor sample choices might result in suboptimal exploration or overlooking important design regions. Additionally, integrating active learning into hardware optimization may require additional computational resources to manage dynamic sample selection. It is important to recognize that while active learning enhances optimization efficiency, its benefits could diminish as fewer informative samples remain available for selection, underscoring the need for a well-balanced exploration-exploitation strategy.

We used active learning in Chapter 5 in conjunction with multi-model Bayesian optimization to balance the exploration-exploitation schemes.

6.5.8 Multi Model Approaches

Multi-model approaches, such as multi-model Bayesian optimization, present an appealing avenue for enriching the process of hardware modeling. These methods leverage the strengths of multiple predictive models to navigate complex design spaces. Multi-model methods are particularly advantageous when dealing with complex relationships between design variables and performance metrics, multi-model approaches aim to capture a more accurate representation of the design landscape. This can lead to improved optimization efficiency by enabling

better exploration and exploitation of the design spaces, ultimately resulting in more optimized hardware designs. However, adopting multi-model strategies requires careful choices in terms of model selection and combination. Effectively coordinating the predictions of different models and managing their interactions while addressing potential computational overhead adds complexity to the optimization process.

Despite these considerations, multi-model approaches hold the potential to achieve valuable insights in hardware optimization. They can mitigate uncertainty associated with single-model predictions and enhance the robustness of the optimization process. The success of multi-model Bayesian optimization relies on the quality and diversity of chosen models, as well as the prudent allocation of computational resources for their training and maintenance. Maintaining a balance between heightened accuracy and increased computational demands remains essential for effectively integrating multi-model strategies into the realm of hardware optimization. We used a multi-model Bayesian approach for modeling various objective functions of the hardware such as power consumption, area utilization, and throughput in Chapter 5.

Transfer Learning

Transfer learning, particularly in the Bayesian optimization framework, is a useful technique for hardware optimization problems. This approach aims to expedite the optimization of new hardware configurations by leveraging insights from previous optimizations. Bayesian optimization's probabilistic nature aligns well with transfer learning, enabling the integration of past optimization knowledge to guide exploration toward promising design choices. This can lead to faster convergence, reduced computational costs, and more informed decision-making in the design space.

On the other hand, challenges arise in implementing transfer learning for hardware optimization. The constraint of *inductive transfer learning*, where source and target tasks share common domains, limits its applicability. While inductive transfer learning offers benefits within shared domains, as demonstrated in Chapter 5, it might struggle when domain characteristics differ between tasks. This choice can lead to sub-optimal knowledge transfer when the source domain's insights aren't entirely suitable for the target domain. As a result, careful selection of source tasks becomes essential to ensure relevant and beneficial knowledge transfer for effective hardware optimization.

Moreover, the focus on inductive transfer learning raises considerations regarding out-of-domain generalization. As this approach confines knowledge transfer to shared domains, it might not address optimization scenarios where domain characteristics significantly deviate.

This could result in missed opportunities to gain information from related but distinct tasks, potentially limiting optimization efficiency in scenarios with dissimilar design spaces. Balancing the advantages of inductive transfer learning with the need for generalization to diverse domains is a crucial aspect in the pursuit of efficient hardware optimization strategies.

6.6 Future Directions

The future trajectory of this research is clearly aimed at addressing the limitations highlighted in Section 6.4. We envision three intertwined hardware design optimization projects for our future work, advanced statistical methods for performance assessment and design space exploration, efficient data management, and transfer learning for adaptive digital architectures. Our objective is to enhance hardware efficiency and expand the frontiers of contemporary digital design. A comprehensive discussion of these projects is presented in Chapter 7.

CHAPTER 7 CONCLUSION AND RECOMMENDATIONS

7.1 Summary of Works and Contributions

Efficient hardware design is one of the most important challenges that researchers and engineers are facing nowadays. Efficient computing platforms are essential to support numerous novel algorithms that are needed for our society. Efficient hardware design is also important because of its socio-economical effect on our lives. Efficient computing platforms not only provide various affordable services to society but also have less carbon footprint and consequently less adverse effects on climate change.

Designing computer hardware is a complex multi-objective problem that deals with multiple parameters and their interactions. In this thesis, we proposed various statistical and machine learning approaches to solve this problem. More specifically, we presented three contributions that deal with the efficient design of hardware for various applications. These contributions tackle *design space exploration* and *performance modeling* as two prominent problems in the field of hardware design. The detailed summary of these contributions is as follows.

In Chapter 3, we used hill climbing and reinforcement learning as the main methods for performing design space exploration. Hill climbing is very efficient in design spaces with lower dimensions, however, in high dimensional data, the number of neighbors to evaluate for finding the steepest ascend becomes challenging. Moreover, the hill climbing approach only works for convex objective functions. This means the hill climbing method may fail to find the global maximum when the objective function is non-convex. On the other hand, reinforcement learning also has its own disadvantages. The most important one is that it requires a large amount of data to train the RL agent. The data requirement of reinforcement learning goes against the main goal of design space exploration; as in design space exploration, we want to minimize the number of evaluations and achieve the best hardware performance. The increasing number of data samples required to perform design space exploration using RL becomes irrational as it gets close to evaluating all possible combinations of parameter sets in the design space. To solve this issue, we proposed using Bayesian optimization in conjunction with Grey Wolf Optimization in the next chapter.

In Chapter 4, we proposed using Bayesian optimization to solve the issues encountered when using reinforcement learning and hill climbing. We have shown that Bayesian optimization is efficient in non-convex objective spaces as opposed to hill climbing. Moreover, the Bayesian optimization method is designed to use fewer samples to create the Bayesian model compared

to reinforcement learning. Another advantage of Bayesian optimization is that it produces a surrogate model of the hardware. We have shown that this model can be used to study inter-dependencies between the design parameters. We explored these inter-dependencies briefly in the form of a Pareto frontier. The main disadvantage of Bayesian optimization is that it is very sensitive to the way that samples are chosen. Biased sampling may help in this case, since we want to choose the samples in the direction of maximizing the objectives. To perform the biased sampling approach for getting suitable samples required to create the Bayesian model, we used the grey wolf optimization method. Since in GWO, the wolves collaborate and communicate to find the optimum points, the explored data samples of wolves are excellent data samples to be fed to a Bayesian model. Combining GWO and BO creates *hybrid* Bayesian models that can perform design space exploration more efficiently compared to both GWO and BO approaches individually. The main disadvantage of a hybrid GWO-BO method is that it still incorporates some randomness because of the random nature of the samples provided by GWO method. This issue is revisited and mitigated in the next chapter.

In Chapter 5, we adapted active learning as a primary sample selection strategy. This means that the Gaussian models of the target objectives collectively choose the next batch of samples for evaluation. We have shown by various experimental results that this method constantly achieves better results in terms of design space exploration compared to other methods proposed in the previous chapter. Furthermore, by showing the fact that the design space exploration of many various tasks are mathematically correlated, we developed a Bayesian transfer learning strategy to transfer the expertise (probabilistic information) acquired in the design space exploration of one task to another task. Moreover, we used Gaussian regression bootstrapping to create simulated data of the hardware and performed performance prediction tests using various regression analyses on them. In all tests, the proposed statistical surrogate model was behaving very closely to the real model. The disadvantage of this method is that the Bayesian models are created assuming the Gaussian distribution of noise for sampling process of the objective function. This led us to using Gaussian processes to create these statistical models. Although Gaussian processes achieve excellent results in our experiments, this assumption might not hold for all hardware models. This means that other types of random processes must be used to address this problem. Some preliminary ideas to solve this issue are proposed in Chapter 7 as the future works.

To conclude, this thesis addresses the problem of efficient hardware design using a multi-disciplinary approach. The proposed approach is inspired by advanced techniques in the fields of hardware design, statistical modeling, and statistical machine learning. Throughout this manuscript, we have shown that one promising way to tackle this problem is the statistical

modeling of the computer hardware architecture. A statistical surrogate model is a useful abstraction that can reveal functional and physical inter-dependencies in the actual model prior to manufacturing it. Ultimately, our proposed contribution can be used to provide a computer-aided design tool that can help the computer hardware industry to design more efficient, affordable, and green computer architectures that provide proper throughput for various modern tasks and algorithms.

7.2 Future Research

Our vision for future works encompasses three main interlinked projects, each addressing critical aspects of hardware design optimization. (i) Firstly, we propose to develop advanced statistical methods for performance evaluation and design space exploration, enabling designers to make informed decisions and uncover hidden design potentials. (ii) Secondly, our focus extends to harnessing statistical methods for efficient on-chip and off-chip data management, streamlining data handling and storage in hardware systems. (iii) Lastly, we envision employing transfer learning techniques to achieve out-of-domain generalization of statistical models for digital hardware design, facilitating the development of adaptive and robust digital architectures. Through the synergistic combination of these projects, we strive to usher in a new era of enhanced hardware efficiency, advancing the boundaries of what is achievable in modern digital hardware design.

The future of this research will therefore be focused on computer architecture, especially focusing on the hardware devices that are optimized for machine learning and 5G/6G communication applications. Thus, the main topics are:

Topic 1: Accurately model hardware performance using multi-objective optimization techniques, studying the Pareto optimal solutions that jointly optimize various hardware-related objective functions such as power efficiency, throughput, and area utilization. The goal is to develop accurate models that enable efficient design space exploration, maximizing various hardware efficiency metrics for specific target applications.

Topic 2: Utilize statistical analysis to enhance on-chip and off-chip data handling. For instance, adapt the hardware to better handle cache misses and branch mispredictions for specific applications e.g. deep learning. Additionally, optimize interactions with off-chip memory, such as RAM, by analyzing the statistics of data movements tailored to applications like deep learning.

Topic 3: Leverage the statistical models developed in Topic 1 and Topic 2 to optimize the design of new hardware architectures that have not been explored previously. For instance,

when a new 6G communication application arises, requiring specific hardware architecture that has never been designed before, use the previously created statistical models to gain insights into the new hardware design space. This objective relies on achieving out-of-domain generalization of statistical models developed in Topic 1 and Topic 2, enabling the transfer of information to the new application domain.

The advancements in statistical machine learning have brought forth new opportunities to evaluate, model, and optimize the digital hardware devices that play a crucial role in our modern society. In this thesis, Bayesian models have been widely utilized to model hardware performance and conduct design space exploration. These Bayesian models are constructed using Gaussian processes [114] to derive closed-form mathematical representations of the hardware. Moreover, they are often integrated with advanced machine learning techniques, such as active learning and transfer learning, to enhance the efficiency and accuracy of these models. In Topic 1, we aim to address a critical limitation of the currently available statistical methods. The existing Bayesian models utilize Gaussian processes, assuming a normal distribution as the prior distribution of the samples. Unfortunately, this assumption may not hold for all hardware applications, leading to improper mathematical modeling of the devices. To overcome this challenge, we propose employing q-Gaussian processes [113], which rely on assuming a q-Gaussian prior distribution. The q-Gaussian distribution generalizes Gaussian processes, enabling the modeling of heavy-tailed as well as bounded support distributions. By using q-Gaussian processes, we can create more accurate statistical models, better suited to capture the complexities and characteristics of the hardware under consideration.

In Topic 2, we propose focusing on optimizing critical performance hotspots. In modern applications, such as deep learning models with billions of parameters, data movement (both on-chip and off-chip) plays a crucial role in performance optimization. Recognizing the statistical fact that certain parameters are less significant in the overall computational scheme is essential for the hardware optimization of such algorithms. Extensive literature on quantization, approximate matrix multiplication, and pruning strategies for deep learning applications supports this observation. Leveraging this phenomenon, we can redesign the branch predictor of the processor or the branch divergence scheduler of GPU devices. Additionally, detecting patterns in data loading can enable us to store the data in a way that promotes cache coherency and reduces cache mispredictions.

In Topic 3, we propose working on a more long-term research endeavor that addresses another significant limitation of the currently available approaches. In recent years, researchers have employed inductive transfer learning for the performance modeling of hardware. However, a drawback of inductive transfer learning is its reliance on the same domain for both the source

and target applications to transfer probabilistic information effectively. The objective is to propose a novel transfer learning strategy capable of performing out-of-domain generalization of the models created in Topic 1 and Topic 2. By achieving this, we can use these models as a starting point for tackling emerging unknown applications in the future.

REFERENCES

[1] "How much does it cost to power data centers?" https://www.sunbirddcim.com/blog/how-much-does-it-cost-power-one-rack-data-center, accessed: 2023-03-03.

[2] Z. Bao and T. Watanabe, "A new approach for circuit design optimization using genetic algorithm," in *2008 International SoC Design Conference*, vol. 01, 2008, pp. I–383–I–386.

[3] V. Krishnan and S. Katkoori, "A genetic algorithm for the design space exploration of datapaths during high-level synthesis," *IEEE Transactions on Evolutionary Computation*, vol. 10, no. 3, pp. 213–229, 2006.

[4] C. Bolchini, P. L. Lanzi, and A. Miele, "A multi-objective genetic algorithm framework for design space exploration of reliable fpga-based systems," in *IEEE Congress on Evolutionary Computation*, 2010, pp. 1–8.

[5] V. Srinivasan, S. Radhakrishnan, and R. Vemuri, "Hardware software partitioning with integrated hardware design space exploration," in *Proceedings Design, Automation and Test in Europe*, 1998, pp. 28–35.

[6] I. D. Anderson and M. A. Khalid, "Sc build: a computer-aided design tool for design space exploration of embedded central processing unit cores for field-programmable gate arrays," *IET Computers & Digital Techniques*, vol. 3, no. 1, pp. 24–32, 2009.

[7] J. Marot, C. Migliaccio, J. Lantéri, P. Lauga, S. Bourennane, and L. Brochier, "Joint design of the hardware and the software of a radar system with the mixed grey wolf optimizer: Application to security check," *Remote Sensing*, vol. 12, no. 18, p. 3097, 2020.

[8] P. Siddavaatam and R. Sedaghat, "Grey wolf optimizer driven design space exploration: a novel framework for multi-objective trade-off in architectural synthesis," *Swarm and Evolutionary Computation*, vol. 49, pp. 44–61, 2019.

[9] S. Prakasam, M. Venkatachalam, and M. Saroja, "Grey wolf optimizer for constrained hardware-software codesign partitioning," *Programmable Device Circuits and Systems*, vol. 8, no. 8, pp. 239–243, 2016.

[10] S. M. Khah, A. S. Mahboob, S. Shahbandegan, and S. H. Zahiri, "Optimal design of a low power, high speed one-bit full adder using multi-objective grey wolf optimizer," in *2020 6th Iranian Conference on Signal Processing and Intelligent Systems (ICSPIS)*, 2020, pp. 1–5.

[11] B. C. Lee and D. M. Brooks, "Accurate and efficient regression modeling for microarchitectural performance and power prediction," *ACM SIGOPS operating systems review*, vol. 40, no. 5, pp. 185–194, 2006.

[12] B. C. Lee and D. M. Brooks, "Illustrative design space studies with microarchitectural regression models," in *2007 IEEE 13th International Symposium on High Performance Computer Architecture*, 2007, pp. 340–351.

[13] C. Dubach, T. Jones, and M. O'Boyle, "Microarchitectural design space exploration using an architecture-centric approach," in *40th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO 2007)*, 2007, pp. 262–271.

[14] W. Jia, K. A. Shaw, and M. Martonosi, "Stargazer: Automated regression-based gpu design space exploration," in *2012 IEEE International Symposium on Performance Analysis of Systems & Software*. IEEE, 2012, pp. 2–13.

[15] K. Sangaiah, M. Hempstead, and B. Taskin, "Uncore rpd: Rapid design space exploration of the uncore via regression modeling," in *2015 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, 2015, pp. 365–372.

[16] Q. Gautier, A. Althoff, P. Meng, and R. Kastner, "Spector: An opencl fpga benchmark suite," in *2016 International Conference on Field-Programmable Technology (FPT)*. IEEE, 2016, pp. 141–148.

[17] J. Meng, L. Yang, X. Peng, S. Yu, D. Fan, and J.-S. Seo, "Structured pruning of rram crossbars for efficient in-memory computing acceleration of deep neural networks," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 68, no. 5, pp. 1576–1580, 2021.

[18] H.-Y. Liu and L. P. Carloni, "On learning-based methods for design-space exploration with high-level synthesis," in *Proceedings of the 50th annual design automation conference*, 2013, pp. 1–7.

[19] G. Singh, J. Gómez-Luna, G. Mariani, G. F. Oliveira, S. Corda, S. Stuijk, O. Mutlu, and H. Corporaal, "Napel: Near-memory computing application performance prediction

via ensemble learning," in *Proceedings of the 56th annual design automation conference 2019*, 2019, pp. 1–6.

[20] C. Lo and P. Chow, "Model-based optimization of high level synthesis directives," in *2016 26th International Conference on Field Programmable Logic and Applications (FPL)*. IEEE, 2016, pp. 1–10.

[21] E. Brochu, V. M. Cora, and N. De Freitas, "A tutorial on bayesian optimization of expensive cost functions, with application to active user modeling and hierarchical reinforcement learning," *arXiv preprint arXiv:1012.2599*, 2010.

[22] A. Mehrabi, A. Manocha, B. C. Lee, and D. J. Sorin, "Bayesian optimization for efficient accelerator synthesis," *ACM Transactions on Architecture and Code Optimization (TACO)*, vol. 18, no. 1, pp. 1–25, 2020.

[23] B. Reagen, J. M. Hernández-Lobato, R. Adolf, M. Gelbart, P. Whatmough, G.-Y. Wei, and D. Brooks, "A case for efficient accelerator design space exploration via bayesian optimization," in *2017 IEEE/ACM International Symposium on Low Power Electronics and Design (ISLPED)*. IEEE, 2017, pp. 1–6.

[24] P. Luong, S. Gupta, D. Nguyen, S. Rana, and S. Venkatesh, "Bayesian optimization with discrete variables," in *Australasian Joint Conference on Artificial Intelligence*. Springer, 2019, pp. 473–484.

[25] H. Ranganathan, H. Venkateswara, S. Chakraborty, and S. Panchanathan, "Deep active learning for image classification," in *2017 IEEE International Conference on Image Processing (ICIP)*. IEEE, 2017, pp. 3934–3938.

[26] A. Siddhant and Z. C. Lipton, "Deep bayesian active learning for natural language processing: Results of a large-scale empirical study," *arXiv preprint arXiv:1808.05697*, 2018.

[27] R. Ibrahim, N. A. Yousri, M. A. Ismail, and N. M. El-Makky, "Multi-level gene/mirna feature selection using deep belief nets and active learning," in *2014 36th Annual International Conference of the IEEE Engineering in Medicine and Biology Society*. IEEE, 2014, pp. 3957–3960.

[28] L. P. Kaelbling, M. L. Littman, and A. W. Moore, "Reinforcement learning: A survey," *Journal of artificial intelligence research*, vol. 4, pp. 237–285, 1996.

[29] M. de Prado, A. Mundy, R. Saeed, M. Denna, N. Pazos, and L. Benini, "Automated design space exploration for optimised deployment of dnn on arm cortex-a cpus," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2020.

[30] A. Ghaffari and Y. Savaria, "Cnn2gate: An implementation of convolutional neural networks inference on fpgas with automated design space exploration," *Electronics*, vol. 9, no. 12, p. 2200, 2020.

[31] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu, "Asynchronous methods for deep reinforcement learning," in *International conference on machine learning*, 2016, pp. 1928–1937.

[32] A. Hosny, S. Hashemi, M. Shalan, and S. Reda, "Drills: Deep reinforcement learning for logic synthesis," in *2020 25th Asia and South Pacific Design Automation Conference (ASP-DAC)*. IEEE, 2020, pp. 581–586.

[33] V. R. Konda and J. N. Tsitsiklis, "Onactor-critic algorithms," *SIAM journal on Control and Optimization*, vol. 42, no. 4, pp. 1143–1166, 2003.

[34] S.-C. Kao, G. Jeong, and T. Krishna, "Confuciux: Autonomous hardware resource assignment for dnn accelerators using reinforcement learning," in *2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 2020, pp. 622–636.

[35] J. Knowles and D. Corne, "On metrics for comparing nondominated sets," in *Proceedings of the 2002 Congress on Evolutionary Computation. CEC'02 (Cat. No.02TH8600)*, vol. 1, 2002, pp. 711–716 vol.1.

[36] D. Van Veldhuizen and G. Lamont, "On measuring multiobjective evolutionary algorithm performance," in *Proceedings of the 2000 Congress on Evolutionary Computation. CEC00 (Cat. No.00TH8512)*, vol. 1, 2000, pp. 204–211 vol.1.

[37] C. Bai, Q. Sun, J. Zhai, Y. Ma, B. Yu, and M. D. Wong, "Boom-explorer: Risc-v boom microarchitecture design space exploration framework," in *2021 IEEE/ACM International Conference On Computer Aided Design (ICCAD)*. IEEE, 2021, pp. 1–9.

[38] S. Liu, F. C. Lau, and B. C. Schafer, "Accelerating fpga prototyping through predictive model-based hls design space exploration," in *Proceedings of the 56th Annual Design Automation Conference 2019*, 2019, pp. 1–6.

[39] M. Awais, H. Ghasemzadeh Mohammadi, and M. Platzner, "Ldax: a learning-based fast design space exploration framework for approximate circuit synthesis," in *Proceedings of the 2021 on Great Lakes Symposium on VLSI*, 2021, pp. 27–32.

[40] D. Paletti, D. Conficconi, and M. D. Santambrogio, "Dovado: An open-source design space exploration framework," in *2021 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*. IEEE, 2021, pp. 128–135.

[41] "Spearmint: Asoftware package for bayesian optimization." https://github.com/HIPS/Spearmint, accessed: 2023-03-03.

[42] "Implementation of some deep learning algorithms." https://github.com/nitishsrivastava/deepnet, accessed: 2023-03-03.

[43] Y. S. Shao, B. Reagen, G.-Y. Wei, and D. Brooks, "Aladdin: A pre-rtl, power-performance accelerator simulator enabling large design space exploration of customized architectures," *ACM SIGARCH Computer Architecture News*, vol. 42, no. 3, pp. 97–108, 2014.

[44] D. Wang, K. Xu, and D. Jiang, "Pipecnn: An opencl-based open-source fpga accelerator for convolution neural networks," in *2017 International Conference on Field Programmable Technology (ICFPT)*. IEEE, 2017, pp. 279–282.

[45] S. I. Venieris and C. S. Bouganis, "fpgaConvNet: Mapping Regular and Irregular Convolutional Neural Networks on FPGAs," *IEEE Transactions on Neural Networks and Learning Systems*, pp. 1–17, 2018.

[46] Y. Umuroglu, N. J. Fraser, G. Gambardella, M. Blott, P. Leong, M. Jahre, and K. Vissers, "Finn: A framework for fast, scalable binarized neural network inference," in *Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*. ACM, 2017, pp. 65–74.

[47] Y. Ma, Y. Cao, S. Vrudhula, and J.-s. Seo, "Optimizing loop operation and dataflow in fpga acceleration of deep convolutional neural networks," in *Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*. ACM, 2017, pp. 45–54.

[48] O. Bilaniuk, S. Wagner, Y. Savaria, and J.-P. David, "Bit-slicing fpga accelerator for quantized neural networks," in *2019 IEEE International Symposium on Circuits and Systems (ISCAS)*. IEEE, 2019, pp. 1–5.

[49] J. Vasiljevic, R. Wittig, P. Schumacher, J. Fifield, F. M. Vallina, H. Styles, and P. Chow, "Opencl library of stream memory components targeting fpgas," in *2015 international conference on field programmable technology (FPT)*. IEEE, 2015, pp. 104–111.

[50] J. Duarte, S. Han, P. Harris, S. Jindariani, E. Kreinar, B. Kreis, J. Ngadiuba, M. Pierini, R. Rivera, N. Tran *et al.*, "Fast inference of deep neural networks in fpgas for particle physics," *Journal of Instrumentation*, vol. 13, no. 07, p. P07027, 2018.

[51] C. Zhang, G. Sun, Z. Fang, P. Zhou, P. Pan, and J. Cong, "Caffeine: Towards uniformed representation and acceleration for deep convolutional neural networks," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2018.

[52] S. Mirjalili, S. M. Mirjalili, and A. Lewis, "Grey wolf optimizer," *Advances in engineering software*, vol. 69, pp. 46–61, 2014.

[53] K. Kandasamy, J. Schneider, and B. Póczos, "Bayesian active learning for posterior estimation," in *Twenty-Fourth International Joint Conference on Artificial Intelligence*, 2015.

[54] W. Rawat and Z. Wang, "Deep convolutional neural networks for image classification: A comprehensive review," *Neural computation*, vol. 29, no. 9, pp. 2352–2449, 2017.

[55] D. Strigl, K. Kofler, and S. Podlipnig, "Performance and scalability of gpu-based convolutional neural networks," in *2010 18th Euromicro Conference on Parallel, Distributed and Network-based Processing*. IEEE, 2010, pp. 317–324.

[56] R. Krishnamoorthi, "Quantizing deep convolutional networks for efficient inference: A whitepaper," *arXiv preprint arXiv:1806.08342*, 2018.

[57] N. Wang, J. Choi, D. Brand, C.-Y. Chen, and K. Gopalakrishnan, "Training deep neural networks with 8-bit floating point numbers," in *Advances in neural information processing systems*, 2018, pp. 7675–7684.

[58] E. Nurvitadhi, G. Venkatesh, J. Sim, D. Marr, R. Huang, J. Ong Gee Hock, Y. T. Liew, K. Srivatsan, D. Moss, S. Subhaschandra *et al.*, "Can fpgas beat gpus in accelerating next-generation deep neural networks?" in *Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*. ACM, 2017, pp. 5–14.

[59] S. Li, K. Sun, Y. Luo, N. Yadav, and K. Choi, "Novel cnn-based ap2d-net accelerator: An area and power efficient solution for real-time applications on mobile fpga," *Electronics*, vol. 9, no. 5, p. 832, 2020.

[60] Intel, "Intel User-Customizable Soc FPGAs," https://www.intel.com/content/dam/www/programmable/us/en/pdfs/literature/wp/wp-01167-custom-arm-soc.pdf, 2019.

[61] T. Feist, "Vivado design suite," *White Paper*, vol. 5, p. 30, 2012.

[62] "Intel quartus prime software," https://www.intel.com/content/www/us/en/products/details/fpga/development-tools/quartus-prime.html, accessed: 2023-03-03.

[63] ONNX, "Open neural network exchange format," 2018. [Online]. Available: https://github.com/onnx/onnx

[64] "hls4ml project current status," https://hls-fpga-machine-learning.github.io/hls4ml/STATUS.html, accessed: 2019-07-13.

[65] U. Aydonat, S. O'Connell, D. Capalija, A. C. Ling, and G. R. Chiu, "An opencl deep learning accelerator on arria 10," in *Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*. ACM, 2017, pp. 55–64.

[66] N. Suda, V. Chandra, G. Dasika, A. Mohanty, Y. Ma, S. Vrudhula, J.-s. Seo, and Y. Cao, "Throughput-optimized opencl-based fpga accelerator for large-scale convolutional neural networks," in *Proceedings of the 2016 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*. ACM, 2016, pp. 16–25.

[67] C. Zhang, P. Li, G. Sun, Y. Guan, B. Xiao, and J. Cong, "Optimizing fpga-based accelerator design for deep convolutional neural networks," in *Proceedings of the 2015 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*. ACM, 2015, pp. 161–170.

[68] Y. Ma, N. Suda, Y. Cao, J.-s. Seo, and S. Vrudhula, "Scalable and modularized rtl compilation of convolutional neural networks onto fpga," in *2016 26th International Conference on Field Programmable Logic and Applications (FPL)*. IEEE, 2016, pp. 1–8.

[69] D. Wang, K. Xu, Q. Jia, and S. Ghiasi, "Abm-spconv: A novel approach to fpga-based acceleration of convolutional neural network inference," in *Proceedings of the 56th Annual Design Automation Conference 2019*. ACM, 2019, p. 87.

[70] K. Wang, Z. Liu, Y. Lin, J. Lin, and S. Han, "Haq: Hardware-aware automated quantization with mixed precision," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2019, pp. 8612–8620.

[71] A. Yazdanbakhsh, A. T. Elthakeb, P. Pilligundla, and F. M. H. Esmaeilzadeh, "Releq: An automatic reinforcement learning approach for deep quantization of neural networks," *arXiv preprint arXiv:1811.01704*, 2018.

[72] I. Grondman, L. Busoniu, G. A. Lopes, and R. Babuska, "A survey of actor-critic reinforcement learning: Standard and natural policy gradients," *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, vol. 42, no. 6, pp. 1291–1307, 2012.

[73] B. Zhang and J. Wei, "Hardware implementation for haze removal with adaptive filtering," *IEEE Access*, vol. 7, pp. 142 498–142 506, 2019.

[74] D. Ngo, S. Lee, G.-D. Lee, and B. Kang, "Single-image visibility restoration: A machine learning approach and its 4k-capable hardware accelerator," *Sensors*, vol. 20, no. 20, p. 5795, 2020.

[75] M. P. Véstias, "A survey of convolutional neural networks on edge with reconfigurable computing," *Algorithms*, vol. 12, no. 8, p. 154, 2019.

[76] V. Dumoulin and F. Visin, "A guide to convolution arithmetic for deep learning," *arXiv preprint arXiv:1603.07285*, 2016.

[77] D. D. Gajski and R. H. Kuhn, "New vlsi tools," *Computer*, no. 12, pp. 11–14, 1983.

[78] Terasic, "DE0-Nano-SoC Kit/Atlas-SoC Kit," de0-nano-soc.terasic.com, 2019.

[79] Terasic, "DE1-SoC Board," de1-soc.terasic.com, 2019.

[80] Nallatech, "Nallatech 510 Acceleration Board," https://www.bittware.com/fpga/510t/, 2019.

[81] H. Van Hasselt and M. A. Wiering, "Reinforcement learning in continuous action spaces," in *2007 IEEE International Symposium on Approximate Dynamic Programming and Reinforcement Learning*. IEEE, 2007, pp. 272–279.

[82] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in neural information processing systems*, 2012, pp. 1097–1105.

[83] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.

[84] S. Shi, Q. Wang, P. Xu, and X. Chu, "Benchmarking state-of-the-art deep learning software tools," in *2016 7th International Conference on Cloud Computing and Big Data (CCBD)*. IEEE, 2016, pp. 99–104.

[85] J. Qiu, J. Wang, S. Yao, K. Guo, B. Li, E. Zhou, J. Yu, T. Tang, N. Xu, S. Song *et al.*, "Going deeper with embedded fpga platform for convolutional neural network," in *Proceedings of the 2016 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*. ACM, 2016, pp. 26–35.

[86] K. Van Moffaert, M. M. Drugan, and A. Nowé, "Scalarized multi-objective reinforcement learning: Novel design techniques," in *2013 IEEE Symposium on Adaptive Dynamic Programming and Reinforcement Learning (ADPRL)*. IEEE, 2013, pp. 191–199.

[87] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[88] C. J. Watkins and P. Dayan, "Q-learning," *Machine learning*, vol. 8, no. 3-4, pp. 279–292, 1992.

[89] E.-G. Talbi and T. Muntean, "Hill-climbing, simulated annealing and genetic algorithms: a comparative study and application to the mapping problem," in *[1993] Proceedings of the Twenty-sixth Hawaii International Conference on System Sciences*, vol. 2. IEEE, 1993, pp. 565–573.

[90] C. Audet and W. Hare, *Derivative-free and blackbox optimization*. Springer, 2017.

[91] P. Zhao and B. Yu, "On model selection consistency of lasso," *The Journal of Machine Learning Research*, vol. 7, pp. 2541–2563, 2006.

[92] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, "A fast and elitist multiobjective genetic algorithm: Nsga-ii," *IEEE transactions on evolutionary computation*, vol. 6, no. 2, pp. 182–197, 2002.

[93] B. C. Schafer, T. Takenaka, and K. Wakabayashi, "Adaptive simulated annealer for high level synthesis design space exploration," in *2009 International Symposium on VLSI Design, Automation and Test*. IEEE, 2009, pp. 106–109.

[94] G. Beltrame, L. Fossati, and D. Sciuto, "Decision-theoretic design space exploration of multiprocessor platforms," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 29, no. 7, pp. 1083–1095, 2010.

[95] E. C. Pinto and C. Doerr, "A simple proof for the usefulness of crossover in black-box optimization," in *International Conference on Parallel Problem Solving from Nature*. Springer, 2018, pp. 29–41.

[96] D. H. Wolpert and W. G. Macready, "No free lunch theorems for optimization," *IEEE transactions on evolutionary computation*, vol. 1, no. 1, pp. 67–82, 1997.

[97] P. Ngatchou, A. Zarei, and A. El-Sharkawi, "Pareto multi objective optimization," in *Proceedings of the 13th International Conference on, Intelligent Systems Application to Power Systems*. IEEE, 2005, pp. 84–91.

[98] J. Bergstra and Y. Bengio, "Random search for hyper-parameter optimization." *Journal of machine learning research*, vol. 13, no. 2, 2012.

[99] H. Xie, L. Zhang, and C. P. Lim, "Evolving cnn-lstm models for time series prediction using enhanced grey wolf optimizer," *IEEE Access*, vol. 8, pp. 161 519–161 541, 2020.

[100] B. Z. Aufa, S. Suyanto, and A. Arifianto, "Hyperparameter setting of lstm-based language model using grey wolf optimizer," in *2020 International Conference on Data Science and Its Applications (ICoDSA)*. IEEE, 2020, pp. 1–5.

[101] A. Mohammadzadeh, M. Masdari, F. S. Gharehchopogh, and A. Jafarian, "Improved chaotic binary grey wolf optimization algorithm for workflow scheduling in green cloud computing," *Evolutionary Intelligence*, pp. 1–29, 2020.

[102] M. Palesi and T. Givargis, "Multi-objective design space exploration using genetic algorithms," in *Proceedings of the tenth international symposium on Hardware/software codesign*, 2002, pp. 67–72.

[103] G. Skolidis, "Transfer learning with gaussian processes," Ph.D. dissertation, 2012.

[104] H. Geng, T. Chen, Y. Ma, B. Zhu, and B. Yu, "Ptpt: physical design tool parameter tuning via multi-objective bayesian optimization," *IEEE transactions on computer-aided design of integrated circuits and systems*, vol. 42, no. 1, pp. 178–189, 2022.

[105] A. Ghaffari and Y. Savaria, "Efficient design space exploration of opencl kernels for fpga targets using black box optimization," *IEEE Access*, vol. 9, pp. 136 819–136 830, 2021.

[106] V. Partovi Nia, A. Ghaffari, M. Zolnouri, and Y. Savaria, "Rethinking pareto frontier for performance evaluation of deep neural networks," *International Conference on Machine Learning (ICML) HAET workshop.*, 2022.

[107] P. D. Kirk and M. P. Stumpf, "Gaussian process regression bootstrapping: exploring the effects of uncertainty in time course data," *Bioinformatics*, vol. 25, no. 10, pp. 1300–1306, 2009.

[108] M. G. Genton, "Classes of kernels for machine learning: a statistics perspective," *Journal of machine learning research*, vol. 2, no. Dec, pp. 299–312, 2001.

[109] K. Hayashi, M. Imaizumi, and Y. Yoshida, "On random subsampling of gaussian process regression: A graphon-based analysis," in *International Conference on Artificial Intelligence and Statistics*. PMLR, 2020, pp. 2055–2065.

[110] "Spec2k: Spec cpu2000 v1.3," https://www.spec.org/cpu2000/, accessed: 2023-01-01.

[111] A. Liaw, M. Wiener *et al.*, "Classification and regression by randomforest," *R news*, vol. 2, no. 3, pp. 18–22, 2002.

[112] R. Tibshirani, "Regression shrinkage and selection via the lasso," *Journal of the Royal Statistical Society: Series B (Methodological)*, vol. 58, no. 1, pp. 267–288, 1996.

[113] M. Bożejko, B. Kümmerer, and R. Speicher, "q-gaussian processes: non-commutative and classical aspects," *Communications in Mathematical Physics*, vol. 185, no. 1, pp. 129–154, 1997.

[114] E. Schulz, M. Speekenbrink, and A. Krause, "A tutorial on gaussian process regression: Modelling, exploring, and exploiting functions," *Journal of Mathematical Psychology*, vol. 85, pp. 1–16, 2018.