Correlated Multi-objective Multi-fidelity Optimization for HLS Directives Design

QI SUN, TINGHUAN CHEN, and SITING LIU, The Chinese University of Hong Kong
JIANLI CHEN, Fudan University
HAO YU, Southern University of Science and Technology
BEI YU, The Chinese University of Hong Kong

31

High-level synthesis (HLS) tools have gained great attention in recent years because it emancipates engineers from the complicated and heavy hardware description language writing and facilitates the implementations of modern applications (e.g., deep learning models) on **Field-programmable Gate Array (FPGA)**, by using high-level languages and HLS directives. However, finding good HLS directives is challenging, due to the time-consuming design processes, the balances among different design objectives, and the diverse fidelities (accuracies of data) of the performance values between the consecutive FPGA design stages.

To find good HLS directives, a novel automatic optimization algorithm is proposed to explore the Pareto designs of the multiple objectives while making full use of the data with different fidelities from different FPGA design stages. Firstly, a non-linear Gaussian process (GP) is proposed to model the relationships among the different FPGA design stages. Secondly, for the first time, the GP model is enhanced as correlated GP (CGP) by considering the correlations between the multiple design objectives, to find better Pareto designs. Furthermore, we extend our model to be a deep version deep CGP (DCGP) by using the deep neural network to improve the kernel functions in Gaussian process models, to improve the characterization capability of the models, and learn better feature representations. We test our design method on some public benchmarks (including general matrix multiplication and sparse matrix-vector multiplication) and deep learning-based object detection model iSmart2 on FPGA. Experimental results show that our methods outperform the baselines significantly and facilitate the deep learning designs on FPGA.

CCS Concepts: • **Hardware** → **Electronic design automation;**

Additional Key Words and Phrases: High-level synthesis, correlated multi-objective optimization, multi-fidelity optimization, design space exploration, Gaussian process

ACM Reference format:
Qi Sun, Tinghuan Chen, Siting Liu, Jianli Chen, Hao Yu, and Bei Yu. 2022. Correlated Multi-objective Multi-fidelity Optimization for HLS Directives Design. *ACM Trans. Des. Autom. Electron. Syst.* 27, 4, Article 31 (March 2022), 27 pages.
https://doi.org/10.1145/3503540

This work is supported by The Research Grants Council of Hong Kong SAR (No. CUHK14209420), The Innovation and Technology Fund (No. PRP/065/20FX), and SmartMore.
Authors' addresses: Q. Sun, T. Chen, S. Liu, and B. Yu, Department of Computer Scinece and Engineering, The Chinese University of Hong Kong, State Key Lab of ASIC & System, Fudan University, Shanghai 200433, China; emails: {qsun, thchen, stliu, byu}@cse.cuhk.edu.hk; J. Chen, School of Microelectronics, Southern University of Science and Technology, Shenzhen 518055, China; email: chenjianli@fudan.edu.cn; H. Yu, School of Microelectronics, Southern University of Science and Technology, Shenzhen, China; email: yuh3@sustech.edu.cn.

1084-4309/2022/03-ART31 $15.00
https://doi.org/10.1145/3503540

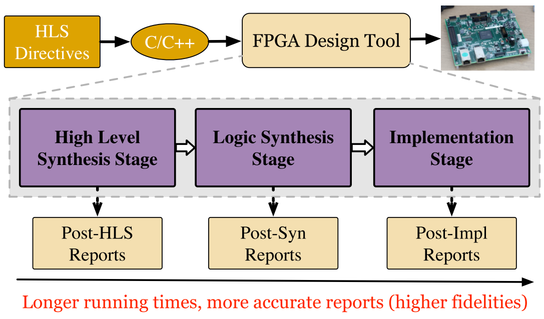
1 INTRODUCTION

The Field-programmable Gate Array (FPGA) design flow is complicated, typically composed of several different steps or phases, including design entry, logic synthesis, and implementation (*aka.*, placement-and-routing). Design entry is to describe the functionalities by the **hardware description languages** (**HDLs**). Logic synthesis turns the HDLs into a design implementation in terms of logic gates. Implementation conducts the placement and routing and generates the bitstream. The workload of the whole flow is heavy and time-consuming. Furthermore, **high-level synthesis** (**HLS**) tools, used as the design entry tools, have made it possible for users who are not experts in writing HDLs to describe their FPGA designs, by translating high-level programming languages (e.g., C/C++) to low-level HDLs, under the guidance of HLS directives. An example of the FPGA design flow relying on HLS is illustrated in Figure 1(a). The HLS directives are embedded into C/C++ source code, as the inputs to the Field-programmable Gate Array (FPGA) design tool.

HLS directives guide the translation process of the high-level language descriptions, in terms of how to parallelize the computations, how to allocate the memory and computation resources, and so on. Given different HLS directive configurations, the final hardware architectures generated from the same high-level language description may vary a lot from each other and therefore have distinctive performance values. In the problem of HLS directives design, the target is to find some HLS directives designs that optimize the design objectives from the entire design space which is composed of candidate directives designs. Some common performance objectives include power, delay, and resource consumption. For better understanding, Figure 1(b) shows an example of HLS directives designs. We need to choose the best factor for each directive to obtain the best performance values. With these advantages, HLS tools have been widely used in many applications, e.g., floating-point computations [2, 20], and deep neural network (DNN) deployments [14, 45].

Several problems still hinder researchers from finding the optimal directives design efficiently. Firstly, it is difficult to find designs that balance the multiple design objectives. For example, reducing system delay demands higher parallelisms which would require more computation cores and have higher resource consumptions, and vice versa. Therefore, optimizing the multiple objectives simultaneously is a multi-objective optimization problem. Secondly, the whole design flow is time-consuming and the analysis reports of these several stages have different fidelities. Later stages can report more accurate analyses, at the cost of longer running times. This kind of multi-stage design problem is also called multi-fidelity design. Besides, the reported results of the three stages in Figure 1(a) and the HLS directives are usually in complicated relationships which make it difficult to map between them. We cannot guarantee whether a design is good or valid in the Implementation stage, though its HLS estimated performance is good. Therefore, we need to predict the quality of the reports at each stage to determine whether we need to run the later FPGA design stages to get more accurate reports.

Some efforts have been made to facilitate the selection of HLS directives. Several analytical/synthesis methods were proposed to analyze the HLS directives, to estimate the performance with no need of running the FPGA design flow for too many directives designs. For the general applications, the directive configurations are analyzed by using an analytic model or a simulator, e.g., Lin-analyzer [57], COMBA [54, 55], polyhedral model [58] and so on. Some proposed to use the dedicated heuristics methods, e.g., lattice [11], clustering [35], divide and conquer [36], and greedy method [30] to guide the exploration of the directive designs. For the deep neural network applications, researchers proposed complicated formulations to simulate the systolic arrays for convolutional operations [39, 46], Spatial/Winograd convolution [52], or several specific code templates [14]. However, these works depend on the accuracy of the analytical models and lack generality or still consume much time to conduct the detailed code analysis and synthesis



(a)

```
comp(int in[10], int out[10]):
  #PRAGMA HLS INLINE={ON, OFF}
  for(i = 0; i < 10; i ++) {
    #PRAGMA HLS UNROLL factor={2,5,10}
    in[i] = out[i];
  }
```

(b)

Fig. 1. (a) The FPGA design flow. C/C++ source code and HLS directives are fed into the design tool. There are three analysis stages and the later stages obtain more accurate reports but consume longer running times. (b) HLS pseudo-codes and directives. The directives are in the boxes, beginning with "#PRAGMA". Each directive has some factors, e.g., ON and OFF of INLINE, and 2, 5, and 10 of UNROLL. Briefly, in this work, our task is to determine the optimal factors for each of these directives.

(*aka.*, profiling) to determine the parameters in the analytical models [14, 39, 46, 52]. Typical parameters include the unit costs of latency, power, and resource consumptions for the given application and FPGA device, which vary significantly under different scenarios. For new applications, new analytic formulations are required, especially for the complicated deep neural networks. These applications challenge the generality of these analytical methods. Besides, the feature of the multiple stages (multi-fidelity) in the FPGA design flow is not considered in these works.

Some model-based works use machine learning algorithms to map from the HLS directives to the performance values, where the complicated FPGA design stages are regarded as black-box functions which can be modeled by machine learning algorithms. Compared to the analytical methods, model-based methods are more flexible and general. The inputs to these models are the feature encodings of the HLS directives and the predicted outputs are the performance values. In these works, the authors collect lots of data to train machine learning models. [27] uses simulated annealing to collect training data, to train a decision tree to guide the exploration of new designs. [21] uses **randomized transductive experimental design** (**RTED**) to draw efficient designs from the design space. [22] guides the FPGA designs according to the known ASIC designs by training a regressor, with the help of [21] as the initialization method. Similarly, [8, 9, 23, 29, 41, 56] propose to use more machine learning algorithms to predict the routing congestions, power, performance and so on, according to the reports and results at various design stages, or the reports of some preliminary analytic models. Some typical algorithms include linear regression, artificial neural networks, and boosting trees. However, huge amounts of real design reports are necessary to guarantee accuracy due to the limited performance of these models. The multiple objectives are usually considered independently, and a machine learning model is built for each objective separately. Besides, the multi-fidelity reports are also not utilized. They use the *post-Implementation* reports at the cost of longer running times or use the *post-HLS* reports at the cost of data accuracies without considering the trade-offs are between running times and data accuracies. To reduce the simulation costs and make full use of the existing reports, recently, **Bayesian optimization** (**BO**) approaches based on the **Gaussian process** (**GP**) have been proposed. However, the multiple objectives are independent of each other [24]. This work is further extended by considering linear multi-fidelity designs [25]. Wider communities have discussed similar tasks, e.g., high-speed adder [13], and multi-core design [19].

However, it is regrettable that some important characteristics of the directive design are ignored. Firstly, the multiple design objectives are in complex correlated relationships. The correlated relationship has been proven to be an important factor in various applications in practical scenarios [6, 7]. Therefore, there exist losses of accuracy in the previous works since they build some independent models to characterize the design objectives. Secondly, the performance values of the three design stages and the directives are in non-linear relationships. It is hard for the designers who implement the high-level descriptions to estimate the performance of the design after placement and routing. Some ignore or evade these complicated relationships by only considering the lowest fidelity (*post-HLS* reports), though they miss some data from the later stages. Unfortunately, to the best of our knowledge, most of the previous methods did not focus on counteracting these challenges explicitly, no matter the analytical methods, or the model-based methods.

In our previous work [40], to help solve these problems, we proposed a novel correlated multi-objective and multi-fidelity GP (CGP) model based on Bayesian optimization. The Bayesian optimization can strike a balance between model accuracies and optimization workloads. Non-linear multi-fidelity models were built to measure the non-linear relationship between the reports of the three design stages and HLS directives. Correlated multi-objective Gaussian process models are proposed as the acquisition functions, to tackle both of the correlated relationships among various design objectives.

Despite that CGP [40] achieved enormous success in modeling the complicated multi-objective and multi-fidelity problem, the shallow structures of Gaussian process models limit the ability to extract information from the input configurations, and bring great challenges to the characterization ability of the kernel functions in the Gaussian process models [48]. Recently, it has been proven that neural networks could automatically discover meaningful representations for the input features by learning multiple layers of highly adaptive basis functions [28, 34, 47–49]. Combining the deep neural networks with GP models can be regarded as the enhancement of the kernel functions, termed as deep kernel functions. The flexibility and automatic calibration provided by the deep kernel functions provide a better performance, with no need for tuning the searching framework for different applications. This technique has achieved more and more attention and applications in the recent few years [34, 47, 49].

In this article, we further extend our CGP method, by introducing the deep kernel functions to learn better feature representations for the configurations and augment the ability of the Gaussian process models. Our contributions are as follows:

— Bayesian optimization is combined with the multi-fidelity and multi-objective optimization as the optimization framework, to trade-off the model accuracies and optimization workloads.
— Non-linear multi-fidelity models are built to measure the non-linear relationship between the reports of the three design stages and HLS directives. Correlated multi-objective Gaussian process models are proposed, to tackle both of the correlated relationships among various design objectives and the implicit and complicated mapping relationships between directives and objective values, to find Pareto configurations accurately.
— Our method is further enhanced to be a deep version **deep CGP** (**DCGP**) by using deep neural networks as the kernel functions, to learn better feature representations flexibly. Therefore the performance of our method can be further improved significantly.
— Three design objectives, power, delay, and resource consumption are considered in this article, thus making the task practical while challenging. We conduct experiments on some public FPGA design benchmarks (including general matrix multiplication and sparse matrix-vector multiplication) and DNN-based object detection model iSmart2. The experimental

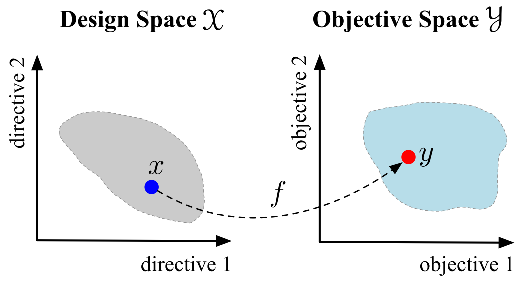


Fig. 2. An example of mapping from design space $\mathcal{X}$ to objective space $\mathcal{Y}$, with two directives and two design objectives (e.g., power and delay). We need to learn a black-box function f to bridge $\mathcal{X}$ and $\mathcal{Y}$, so as to simulate the FPGA design tool.

results show the outstanding performance of our algorithms on the DNN applications and the related applications. The found Pareto designs cover the optimal designs for various design objectives, i.e., power, delay, and resource consumption, and strike a good balance between these objectives.

The rest of our article is organized as follows. Section 2 provides some preliminaries about the multi-objective, multi-fidelity optimizations, Gaussian process, Bayesian optimization, and an overview of our method. Section 3 presents the design space of the HLS directive optimization problem. Section 4 introduces our non-linear correlated multi-objective GP (CGP) model and the deep version (DCGP) model. Section 5 shows our overall optimization flow. Section 6 conducts several experiments to validate our methods, followed by conclusions in Section 7.

2 PRELIMINARIES

In selecting an optimal HLS directive design, our target is finding a configuration of directives in the space of all directive configurations (named *design space* or *configuration space* in our context) $\mathcal{X}$ which have the optimal performance values in the *objective space* (or *value space*) $\mathcal{Y}$. The design space $\mathcal{X}$ is constructed by enumerating possible HLS directives to be used in the high-level language descriptions. In $\mathcal{X}$, each configuration can be represented as a feature vector $\boldsymbol{x}$. The details on directive encoding are in Section 3.2. The objective space $\mathcal{Y}$ is composed of the performance values of the designs given HLS directive configurations. $\mathcal{Y}$ is not known unless we run all of the configurations with the FPGA design tool. Our target is to achieve the configurations with the best performance with no need of knowing the whole objective space $\mathcal{Y}$. In the rest of this article, the three FPGA design stages are shorted as *hls*, *syn*, and *impl*.

2.1 Bayesian Optimization

Bayesian optimization (**BO**) is an efficient and widely-used framework [15, 43] to solve global optimization problems, e.g., optimizing analog circuits [26]. For an optimization problem with a black-box objective function f, e.g., power consumption, whose concrete form is unknown, the target of Bayesian optimization is to find a configuration point $\boldsymbol{x} \in \mathcal{X}$ which has the optimal objective value in the objective space $\mathcal{Y}$ by conducting a limited number of trials or evaluations. A two-dimensional example of the mapping from the design space to the objective space is shown in Figure 2.

In Bayesian optimization, firstly, a set of initial configurations is randomly sampled from the design space $\mathcal{X}$ and passed into the FPGA design tools to get the performance values. These initial data are used to build a *surrogate model* to mimic the objective function. Secondly, the BO algorithm iteratively selects a new configuration from the design space for evaluation under the guidance of an *acquisition function* and then updates the *surrogate model* accordingly. Finally, the optimal HLS directive configuration is the best one explored by the BO algorithm in the optimization process. Three essential elements in Bayesian optimization are as follows:

(1) Surrogate model to optimize the black-box objective function $f : \mathcal{X} \rightarrow \mathcal{Y}$, BO learns a probabilistic surrogate model to predict the function value and quantifies the uncertainty of the predictions. A commonly used surrogate model is the GP model.

(2) Acquisition function $\alpha(\cdot)$ is used as a score function to evaluate the utility of a candidate point $\boldsymbol{x} \in \mathcal{X}$ with respect to finding the optimums of the optimization problem. The acquisition function should balance the exploitation of already-sampled configurations and the exploration of un-sampled configurations in the design space. It is built on the already-sampled configurations and utilizes this prior knowledge (i.e., exploitation) to evaluate the un-sampled configurations (i.e., exploration). There are some popular acquisition functions, such as **expected improvement** (**EI**), **upper confident bound** (**UCB**), and **entropy search** (**ES**) [26].

(3) Optimization procedure iteratively samples a configuration from $\mathcal{X}$ based on the acquisition function $\alpha(\cdot)$ in each optimization step and updates the surrogate model accordingly.

2.2 Multi-objective Optimization

In the optimization problem of HLS directives, there are multiple objectives to be minimized, e.g., power, delay, and consumption of various types of resources. No matter whether the designers clearly emphasize the single-objective or multi-objective in their problems or not, these multiple objectives should always be considered to guarantee the system's performance. Without loss of generality, our goal is to minimize a group of objectives $f^1(\boldsymbol{x}), f^2(\boldsymbol{x}), \ldots, f^M(\boldsymbol{x}), \forall \boldsymbol{x} \in \mathcal{X}$. Denote the objective values of the $\boldsymbol{x}$ as $\boldsymbol{f}(\boldsymbol{x}) = [f^1(\boldsymbol{x}), f^2(\boldsymbol{x}), \ldots, f^M(\boldsymbol{x})]^\top$. These M objectives would possibly conflict with each other. Finding one solution that minimizes all of these objectives simultaneously is difficult. Practically, to strike a balance between these objectives, we want to identify the Pareto-optimal set.

Definition 1 (Pareto optimality). In an M-dimension minimization problem, an objective vector $\boldsymbol{f}(\boldsymbol{x})$ is said to dominate $\boldsymbol{f}(\boldsymbol{x}')$ if

$$\begin{aligned} &\forall i \in [1, M], f^i(\boldsymbol{x}) \leq f^i(\boldsymbol{x}') \text{ and} \\ &\exists j \in [1, M], f^j(\boldsymbol{x}) < f^j(\boldsymbol{x}'). \end{aligned} \tag{1}$$

A point $\boldsymbol{x}$ is Pareto-optimal if there is no other $\boldsymbol{x}'$ in design space satisfying that $\boldsymbol{f}(\boldsymbol{x}')$ dominates $\boldsymbol{f}(\boldsymbol{x})$. In the whole design space, the set of points that are not dominated by others is called the Pareto-optimal set, denoted as $\mathcal{Y}^* \in \mathcal{Y}$. For the Pareto-optimal designs, there does not exist an alternative choice that can improve every objective without sacrificing others. In multi-objective optimization problems, the realistic and accurate goal is to identify the Pareto-optimal set containing all the Pareto-optimal directive configurations. In the previous works on HLS optimization, the M objective functions are solved independently [25], while in this article we proposed to combine them together by a correlation method.

2.3 Multi-fidelity Optimization

Definition 2 (Fidelity). Fidelity refers to the degree to which a model reproduces the state of a real-world project or application. It is therefore a measure of the realism of the model. Straightforwardly, lower fidelity means that the model has lower data accuracy and the higher fidelity is more accurate.

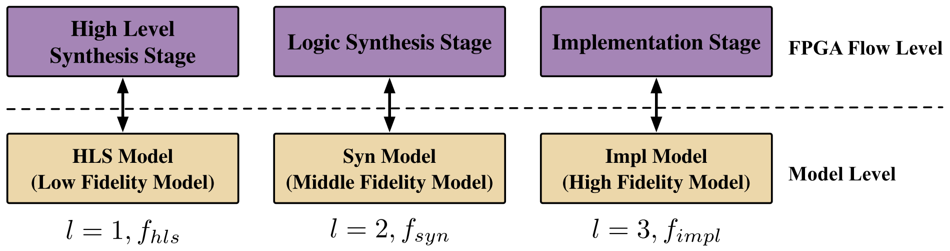

High Level Synthesis Stage
Logic Synthesis Stage
Implementation Stage
FPGA Flow Level
HLS Model (Low Fidelity Model)
Syn Model (Middle Fidelity Model)
Impl Model (High Fidelity Model)
Model Level
$l = 1, f_{hls}$
$l = 2, f_{syn}$
$l = 3, f_{impl}$

Fig. 3. The correspondence relationships between the design stages in the FPGA design flow and the multiple models and fidelities in the model level.

Definition 3 (Multi-fidelity Model). For a multi-fidelity problem, each fidelity $l \in \{1, 2, \ldots, L\}$ corresponds to a objective function $f_l(\boldsymbol{x})$. The multi-fidelity model can be defined as

$$f_{l+1}(\boldsymbol{x}) = z(f_l(\boldsymbol{x}), \boldsymbol{x}), \tag{2}$$

where $z(\cdot)$ is an aggregation function. In our context, $l = 1$ means *hls*, $l = 2$ means *syn*, and $l = 3$ means *impl*.

We need to define three objective functions for the three stages, i.e., $\{f_{hls}, f_{syn}, f_{impl}\}$. As shown in Figure 1(a), the later stages return more accurate reports, at the cost of consuming longer running times and more computation resources. Therefore, the objective functions for the later stages with more accurate reports would have higher fidelities. The correspondence relationships are shown in Figure 3. There are three models, the HLS model, the Syn model, and the Impl model corresponding to the HLS stage, the Syn stage, and the Impl stage, respectively. That is why it is called multi-fidelity optimization.

In the Bayesian optimization, we define three surrogate models to mimic the objective functions for the three stages, and three acquisition functions to evaluate the utilities of new configurations on these three surrogate models respectively. For convenience, in this article, we treat the design *stage* of FPGA design flow and model *fidelity* of algorithms as equivalent and do not distinguish between them, e.g., the lower fidelity implicitly means the lower design stage and vice versa.

2.4 Gaussian Process Regression

GP regression [31] is a flexible method to model the objective function, which is specified by a mean function $\mu(\boldsymbol{x})$ and a covariance function $k(\boldsymbol{x}, \boldsymbol{x}')$ of the objective $f(\boldsymbol{x})$ as follows:

$$\begin{aligned} \mu(\boldsymbol{x}) &= \mathbb{E}[f(\boldsymbol{x})], \\ k(\boldsymbol{x}, \boldsymbol{x}') &= \mathbb{E}[(f(\boldsymbol{x}) - \mu(\boldsymbol{x}))(f(\boldsymbol{x}') - \mu(\boldsymbol{x}'))]. \end{aligned} \tag{3}$$

The mean function $\mu(\boldsymbol{x})$ provides the prior estimations of the objective value for input $\boldsymbol{x}$, and typically a constant mean function $\mu(\boldsymbol{x}) = \mu_0$ is widely used. As to the covariance function $k(\boldsymbol{x}, \boldsymbol{x}')$, the common form is the squared exponential function of $\boldsymbol{x}$:

$$k(\boldsymbol{x}, \boldsymbol{x}') = \lambda^2 \exp\left(-\frac{1}{2}(\boldsymbol{x} - \boldsymbol{x}')^\top \boldsymbol{\Lambda}(\boldsymbol{x} - \boldsymbol{x}')\right), \tag{4}$$

where $\boldsymbol{\Lambda} = \mathrm{diag}(\lambda_1^{-2}, \lambda_2^{-2}, \ldots, \lambda_D^{-2})$ is the diagonal length scale matrix, and λ^2 is used to scale the variance of the model.

Define a known single-objective training set $\{\mathcal{X}, \mathcal{Y}\}$, where $\mathcal{X} = \{\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_n\}$ is a set of directive configurations and $\mathcal{Y} = \{y_1, y_2, \ldots, y_n\}$ is the corresponding single-objective value set. Assume that the objective function $f(\boldsymbol{x})$ is influenced by the independent and identical zero-mean Gaussian noise $\epsilon_e \sim \mathcal{N}(0, \sigma_e^2)$. Therefore, we have the relationship between directive configuration and its corresponding performance $y_i = f(\boldsymbol{x}_i) + \epsilon_e$, with $i = 1, \ldots, n$.

For a newly sampled configuration $\boldsymbol{x}^*$ and its corresponding objective function f^*, the joint distribution between f^* and the data set $\mathcal{Y}$ which is already sampled in previous steps is defined as follows:

$$p(\mathcal{Y}, f^*) = \mathcal{N}\left(\begin{bmatrix} \boldsymbol{\mu}_0 \\ \mu_0 \end{bmatrix}, \begin{bmatrix} \boldsymbol{K}(\mathcal{X}) + \sigma_e^2 \boldsymbol{I} & \boldsymbol{k}(\mathcal{X}, \boldsymbol{x}^*) \\ \boldsymbol{k}^\top(\mathcal{X}, \boldsymbol{x}^*) & k(\boldsymbol{x}^*, \boldsymbol{x}^*) \end{bmatrix}\right), \tag{5}$$

where $\boldsymbol{k}(\mathcal{X}, \boldsymbol{x}^*)$ is a vector of covariance values between $\boldsymbol{x}^*$ and all of the configurations in $\mathcal{X}$, and $\boldsymbol{K}(\mathcal{X})$ is the intra-covariance matrix among configurations in $\mathcal{X}$, i.e., $\boldsymbol{K}(\mathcal{X})_{i,j} = k(\boldsymbol{x}_i, \boldsymbol{x}_j)$ with $\boldsymbol{x}_i, \boldsymbol{x}_j \in \mathcal{X}$. According to the Bayes' theorem, the posterior distribution is obtained by

$$p(f^* | \mathcal{Y}) = \frac{p(\mathcal{Y}, f^*)}{\int p(\mathcal{Y}, f^*) \mathrm{d} f^*} = \mathcal{N}(\mu(\boldsymbol{x}^*), \Sigma(\boldsymbol{x}^*)), \tag{6}$$

with

$$\begin{aligned} \mu(\boldsymbol{x}^*) &= \mu_0 + \boldsymbol{k}^\top(\mathcal{X}, \boldsymbol{x}^*)[\boldsymbol{K}(\mathcal{X}) + \sigma_e^2 \boldsymbol{I}]^{-1}(\mathcal{Y} - \boldsymbol{\mu}_0), \\ \Sigma(\boldsymbol{x}^*) &= k(\boldsymbol{x}^*, \boldsymbol{x}^*) - \boldsymbol{k}^\top(\mathcal{X}, \boldsymbol{x}^*)[\boldsymbol{K}(\mathcal{X}) + \sigma_e^2 \boldsymbol{I}]^{-1} \boldsymbol{k}(\mathcal{X}, \boldsymbol{x}^*). \end{aligned} \tag{7}$$

Before the posterior is calculated, the hyper-parameters $\boldsymbol{\Lambda}$, λ, and σ_e need to be determined by maximum likelihood estimation as follows:

$$\max_{\boldsymbol{\Lambda}, \lambda, \sigma_e} \quad (\mathcal{Y} - \boldsymbol{\mu}_0)^\top (\boldsymbol{K}(\mathcal{X}) + \sigma_e^2 \boldsymbol{I})^{-1} (\mathcal{Y} - \boldsymbol{\mu}_0) + \log |\boldsymbol{K}(\mathcal{X}) + \sigma_n^e \boldsymbol{I}|, \tag{8}$$

which can be handled by gradient-based methodologies.

2.5 Overview of Our Method

In this article, a correlated multi-objective multi-fidelity deep kernel learning GP-based Bayesian Optimization algorithm is proposed to explore the Pareto solutions of HLS directives. The main techniques include the construction of design space, surrogate models and acquisition functions, and optimal configuration selection. In constructing design space, HLS directives are transformed into numerical vectors to perform GP-based Bayesian optimization. Besides, a pruning method is proposed to shrink design space so that the downstream GP-based Bayesian Optimization can explore design space more efficiently. In constructing surrogate models, we combine deep kernel learning with GP to learn better feature representations flexibly. In constructing acquisition functions, we use the expected improvement of Pareto hyper-volume as a metric to select the most representative configuration to run the FPGA design tool. Then performance values obtained from the FPGA design tool and the corresponding configuration are used to extend the dataset and update the models. All techniques mentioned above are used to achieve more efficient design space exploration. The brief optimization flow is shown in Figure 4. In Section 5, the overall detailed optimization flow and algorithm framework are provided to solve the optimization problems of HLS directives.

3 HLS DIRECTIVE DESIGN SPACE

To construct the design space $\mathcal{X}$, we enumerate the designs, conduct the tree-based design space pruning to remove infeasible configurations and encode the directive configurations as feature vectors.

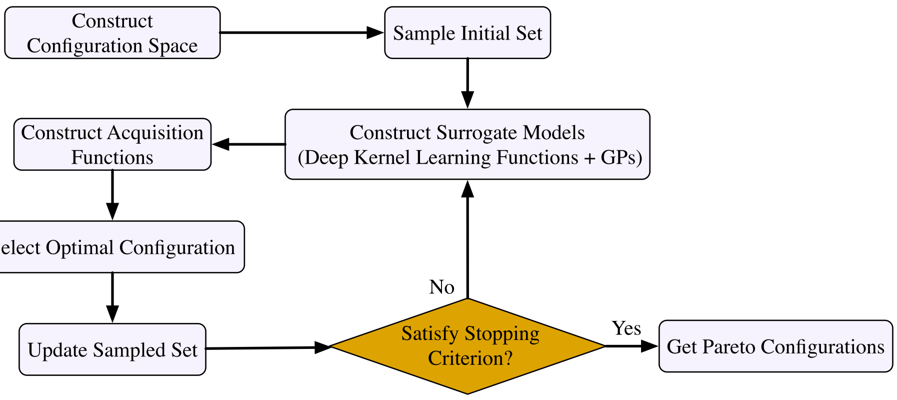
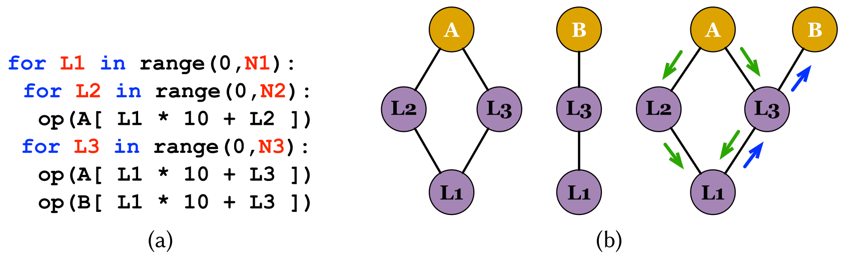

Construct Configuration Space
Sample Initial Set
Construct Surrogate Models (Deep Kernel Learning Functions + GPs)
Construct Acquisition Functions
Select Optimal Configuration
Update Sampled Set
Satisfy Stopping Criterion?
No
Yes
Get Pareto Configurations

Fig. 4. The brief flow of our method.



```
for L1 in range(0,N1):
 for L2 in range(0,N2):
  op(A[ L1 * 10 + L2 ])
 for L3 in range(0,N3):
  op(A[ L1 * 10 + L3 ])
  op(B[ L1 * 10 + L3 ])
```

Fig. 5. An example of tree-based design space pruning method. (a) Code with three loops and two arrays. L1 is the outer loop. L2 and L3 are the inner parallel loops. (b) Trees of array A and B, and the merged tree.

3.1 Design Space and Tree-based Space Pruning

Some HLS directives are widely used, including pipelining, loop unrolling, array partitioning, and so on. These directives are especially popular in DNN-based applications which are composed of many dense matrix operations. Some typical dense matrix operations include fully connected operations, convolutional operations, and **general matrix multiplication** (GEMM). In general applications, the codes are composed of several for-loops, some arrays, and related computations. The design space can be generated by direct permutations and combinations of directives.

However, some directives are conflicting and some are obviously non-optimal, especially for loop unrolling and array partitioning. Infeasible configurations may increase optimization workloads. For example, for an array used in a for-loop, if the array partitioning factor is less than the loop unrolling factor, this loop may not be unrolled successfully because the visits to the array are limited by the partitioning factor [42]. If the array partitioning factor is greater than the loop unrolling factor, more memory resources are consumed without increasing the system parallelism. Under this circumstance, compatible directives and factors are the best. A tree-based design space pruning method is proposed to help solve this problem, as shown in Algorithm 1. The inputs are high-level language descriptions and a file that indicates which directives are to be analyzed.

Figure 5 shows an example. There are two arrays A and B, and three loops L1, L2, and L3 in Figure 5(a). Two trees are built for A and B, with arrays as root nodes and loops as non-root nodes, as shown in Figure 5(b). The outer loop L1 is the leaf node and nested loops L2 and L3 are non-leaf nodes. These two trees are merged since they share some common nodes L3 and L1. In each

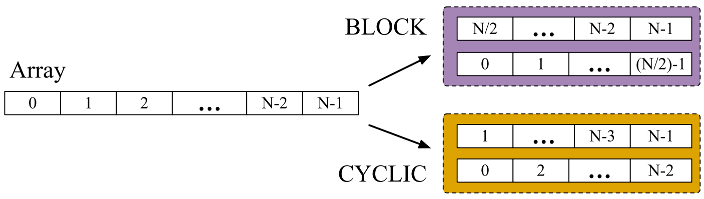


Fig. 6. Two types of array partitioning, CYCLIC, and BLOCK. The original array is stored in a continuous memory space. In this example, the partitioning factor is 2. Both the CYCLIC and BLOCK will partition the array into two separate memory spaces. CYCLIC creates smaller arrays by interleaving elements from the original array. BLOCK creates smaller arrays from consecutive blocks of the original array.

ALGORITHM 1: The Pseudo-code of The Tree-based Pruning Method

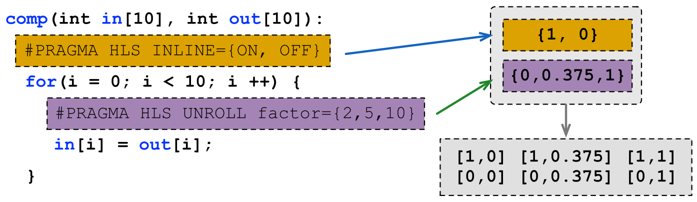
```
1: Inputs: High-level programming language source code, and a directive file;
2: Outputs: Pruned design space $\mathcal{X}$, initially $\mathcal{X} \leftarrow \emptyset$;
3: Construct a tree for each array, with itself as the root node and related loops as children nodes;
4: Merge trees with common nodes, denote the set of trees as $\mathcal{T}$;
5: for all tree $t_i \in \mathcal{T}$ do
6:     for root (array) node $a_j$ in $t_i$ do
7:         for partitioning factor $f_k$ of $a_j$ do
8:             Assign $f_k$ to $a_j$;
9:             Assign a unrolling factor to each loop node in $t_i$;
10:            Backtrack from leaf nodes, assign partitioning factors to array nodes in $t_i$, except $a_j$;
11:        end for
12:        Record feasible configurations of $a_j$ as set $C_j$;
13:        $\mathcal{X} \leftarrow \mathcal{X} \cup C_j$;
14:    end for
15: end for
16: Traverse $\mathcal{X}$ and remove repeated configurations;
17: return Pruned design space $\mathcal{X}$.
```

tree, the factor of each child node is determined by its parents. Two types of array partitioning are considered here, CYCLIC and BLOCK, as shown in Figure 6.

If we partition A with type CYCLIC, then we will assign unrolling factors for L2 and L3. But we will not unroll L1. In other words, the unrolling factor is 1, because L1 is incompatible with CYCLIC partitioning of A and unrolling of L2 and L3. After that, we will backtrack from L1 to assign CYCLIC partitioning factors to B because A and B are in the same loop L3 and their partitioning types should be the same. In the tree, they are connected by the common node L3.

If we partition array A with type BLOCK, we will set the unrolling factors of L2 and L3 as 1. The reason is that the unrolling of these two loops is incompatible with the BLOCK of A. But we can unroll loop L1 successfully because they are compatible. After that, we will backtrack from L1 to B, to partition array B with type BLOCK.

In this process, all factors will be checked whether they are compatible, and more domain knowledge can be used here if wanted. All of the compatible configurations are added into a configuration set C_A belonging to A. After identifying C_A and adding C_A into $\mathcal{X}$, we can conduct the same configuration assignment process starting at array B in the merged tree. Note that finally, we will



```
comp(int in[10], int out[10]):
 #PRAGMA HLS INLINE={ON, OFF}
  for(i = 0; i < 10; i ++) {
    #PRAGMA HLS UNROLL factor={2,5,10}
     in[i] = out[i];
  }
```

Fig. 7. An example of directive encoding. In this example, there are two HLS directives and six configurations in the design space.

traverse $\mathcal{X}$ again to remove repeated configurations. The invalid and incompatible directive configurations are pruned with the tree-based method, which also eases the optimization task. For the applications with extremely large design spaces, sampling techniques can be adopted, e.g., Monte Carlo-based sampling with reparameterization trick [50, 51]. Therefore, our proposed method can be used to explore large design spaces efficiently.

3.2 Encoding of Directive Configurations

The GP, as a typical machine learning model, can only work on numerical values. Therefore, it is necessary to transform the non-numerical design parameters, such as unroll and inline, into numerical arrays. The TRUE/FALSE factors are represented as 0 or 1 directly. The directives which have several factors are represented as normalized features, e.g., three factors $\{2, 5, 10\}$ are encoded as $\{0, 0.375, 1\}$. The normalization can adjust all features to the same scale and allows for a more uniform influence for all weights and faster convergence on learning [5]. If the partitioning factor 2 has good performances, we will conjecture that 5 is better than 10 because 5 is closer to 2. Figure 7 shows an example. More directives with factors are included in the experiments, e.g., pipelining and array partitioning. The final feature vector for a code segment is the concatenations and combinations of features of all the directives in this segment.

4 CORRELATED MULTI-OBJECTIVE MULTI-FIDELITY MODELS AND DEEP KERNEL FUNCTIONS

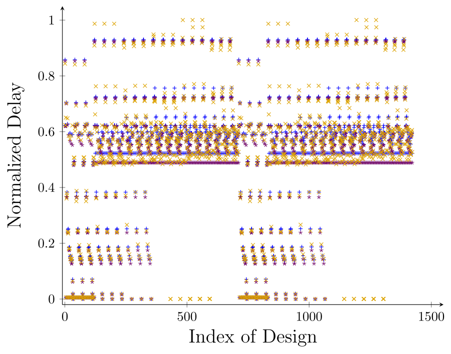
In this section, non-linear multi-fidelity models and correlated multi-objective models are described in Sections 4.1 and 4.2 are enhanced by deep kernel functions in Section 4.4.

4.1 Non-linear Multi-Fidelity Model

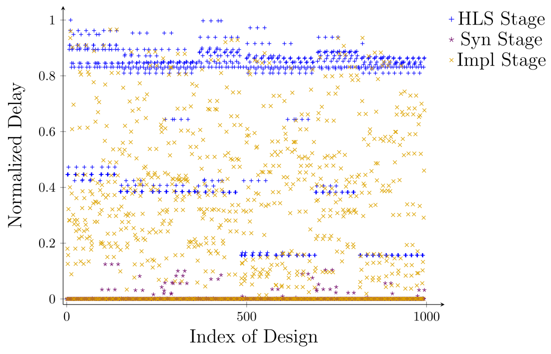
Traditionally, in HLS directive designs, the relationship among the multiple stages (fidelities) is assumed to be linear. For example, in [25], higher fidelity estimates scale the lower fidelity output by a factor and add an independent GP to model the remaining differences. However, it is not suitable and weak for the applications where these three FPGA design stages exhibit strong complicated correlations. Therefore, non-linear models are proposed in this article to further exploit the corresponding non-linear relationships between the low- and high-fidelity objective functions. The non-linear model can be formulated as Equation (9).

$$f_{i+1}(\boldsymbol{x}) = z(f_i(\boldsymbol{x}), \boldsymbol{x}) + f_e(\boldsymbol{x}), \forall i \in \{1, \ldots, L-1\}, \tag{9}$$

where $z(\cdot)$ is the non-linear function and is modeled by a GP model in this article, and $f_e(\boldsymbol{x})$ is the error term which is also defined as a GP model. The outputs $f_i(\boldsymbol{x})$ of the early stage (low Fidelity) model are concatenated with the directive encoding features $\boldsymbol{x}$ as the input features to the later stage (high fidelity) GP model.



(a)



(b)

Fig. 8. Normalized delay values of the three fidelities. The X-axis is the index of the design. We assign indices for these designs according to their directive values in increasing order. (a) GEMM (general matrix multiplication). (b) SPMV_ELLPACK (sparse matrix-vector multiplication using the ELLPACK format).

Delay values of two benchmarks are shown in Figure 8 as examples to illustrate the complex non-linear relationships among the three fidelities. In the GEMM, delay values of the configurations in the three fidelities are highly overlapping. For the sparse matrix-vector multiplication using the ELLPACK format (SPMV_ELLPACK), delay values in the three fidelities show high divergences. The high divergences of various applications make it hard to regress the relationships accurately by using traditional linear models. Obviously, using non-linear models is a wise and general choice to handle various applications.

4.2 Correlated Multi-Objective Model

To learn and measure the Pareto set for the multi-objective optimization problem, we introduce the **expected improvement of Pareto hyper-volume** (**EIPV**) [37] and define it as the acquisition function. Firstly, we will clarify the concept of the expected improvement of Pareto hyper-volume. Secondly, we will define the probability model and compute the value of the expected improvement.

Assume that in current optimization step $t + 1$, we already have a Pareto-optimal set $\mathcal{D} = \{\mathcal{X}^*, \mathcal{Y}^*\}$, with $\mathcal{X}^* = \{\boldsymbol{x}_s\}_{s=1}^t$, and $\mathcal{Y}^* = \{\boldsymbol{y}_s\}_{s=1}^t$. Note that $\mathcal{D}$ is the Pareto-optimal set of the designs explored in the previous t steps. A virtual configuration point $\boldsymbol{v}_{ref} \in \mathbb{R}^M$ is defined as the reference point, which is dominated by $\mathcal{Y}^*$, i.e., $\boldsymbol{y}_s \succeq \boldsymbol{v}_{ref}$[1] for $\forall \boldsymbol{y}_s \in \mathcal{Y}^*$. $\boldsymbol{v}_{ref}$ does not have physical meanings and is only for the ease of computations. In the experiments, we can directly assign extremely large values which usually do not occur in practical scenarios to $\boldsymbol{v}_{ref}$, e.g., 100 W for power. The Pareto hyper-volume with respect to $\boldsymbol{v}_{ref}$ in the objective space is defined as Equation (10).

$$\mathrm{PV}_{\boldsymbol{v}_{ref}}(\mathcal{Y}^*) = \int_{\mathbb{R}^M} \mathbb{I}[\boldsymbol{y} \succeq \boldsymbol{v}_{ref}] \left[1 - \prod_{\boldsymbol{u} \in \mathcal{Y}^*} \mathbb{I}[\boldsymbol{u} \nsucceq \boldsymbol{y}]\right] \mathrm{d}\boldsymbol{y}, \tag{10}$$

where $\mathbb{I}(\cdot)$ is the indicator function, which outputs 1 if its argument is true and 0 otherwise. This equation measures the volume of the objective space composed of configurations, which dominate $\boldsymbol{v}_{ref}$ but are dominated by at least one configuration in $\mathcal{Y}^*$. The greater the volume is, the better the Pareto set is.

[1]"$\succeq$" denotes "dominate". In this minimization problem, its numerical meaning is "$\leq$" as shown in Equation (1).

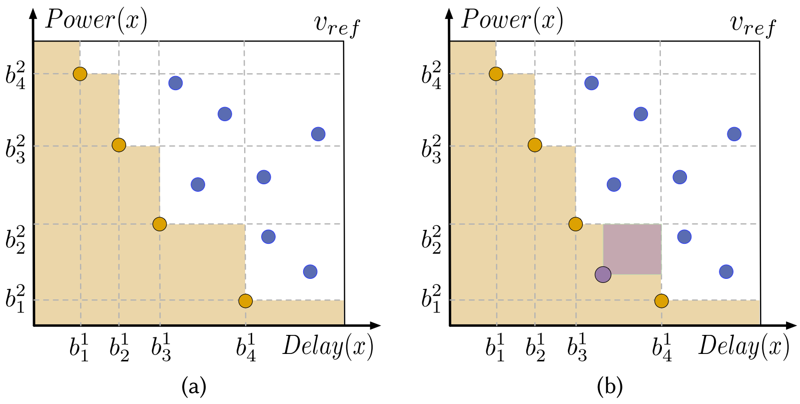


Fig. 9. An example of minimizing power and delay. The objective space is divided into cells according to the locations of the currently found Pareto set. (a) Orange points are Pareto points and blue points are dominated. Blank cells are dominated while light yellow cells are not. The volume of the blank cells is the current Pareto hyper-volume. (b) Purple point is predicted to be the Pareto-optimal configuration and the light purple cell is the corresponding expected improvement of Pareto hyper-volume.

Greedily, in each operation step, we want to sample a configuration, which can lead to the highest expected improvement of the Pareto hyper-volume. Here the "expected" comes from the uncertainty information of the predicted performance values of GP models. We will estimate the expected improvements for the un-sampled configurations and select the configuration which leads to the largest expected improvement. The expected improvement is defined as Equation (11).

$$\mathrm{EIPV}(\boldsymbol{x}_{t+1}|\mathcal{D}) = \mathbb{E}_{p(\boldsymbol{y}(\boldsymbol{x}_{t+1})|\mathcal{D})}\left[\mathrm{PV}_{\boldsymbol{v}_{ref}}(\mathcal{Y}^* \cup \boldsymbol{y}(\boldsymbol{x}_{t+1})) - \mathrm{PV}_{\boldsymbol{v}_{ref}}(\mathcal{Y}^*)\right]. \tag{11}$$

We can decompose the whole objective space into grid cells to simplify the integration of Equation (10), as shown in Figure 9(a). The decomposition is according to the locations of the found Pareto-optimal configurations in the objective space. The corresponding objective values at these two axes are b_i^1 and b_i^2. We denote the non-dominated cells as $\mathcal{C}_{\mathrm{nd}}$. Then Equation (11) is simplified as Equation (12), where $\Delta_C(\boldsymbol{x})$ is the volume of cell $C \in \mathcal{C}_{\mathrm{nd}}$.

$$\mathrm{EIPV}(\boldsymbol{x}_{t+1}|\mathcal{D}) = \sum_{C \in \mathcal{C}_{\mathrm{nd}}} \Delta_C(\boldsymbol{x}) = \sum_{C \in \mathcal{C}_{\mathrm{nd}}} \int_C \mathrm{PV}_{\boldsymbol{v}_c}(\boldsymbol{y}) p(\boldsymbol{y}|\mathcal{D}) \mathrm{d}\boldsymbol{y}. \tag{12}$$

In Figure 9(b), the purple node maximizes the expected improvement and therefore, it is the Pareto design to select in this step.

Now we have clarified the concept of the expected improvement of Pareto hyper-volume. The next step is to define the probability model $p(\boldsymbol{y}|\mathcal{D})$ and to further deduce the concrete form of the expected improvements. In previous works [24, 25], the multiple design objectives are predicted via several independent Gaussian process models, though in real applications, they are usually correlated. For example, to reduce system delay, we may want to increase the system parallelism which means we will consume much more on-chip resources, e.g., LUTs. Therefore, delay and resource consumption are negatively correlated. But power and resource consumption are positively correlated since instantiating more on-chip resources would increase power consumption simultaneously.

In this article, $p(\boldsymbol{y}|\mathcal{D})$ is modeled as a correlated multi-objective GP model [4], as shown in Equation (13).

$$p(\boldsymbol{y}|\mathcal{D}) = \mathcal{N}(y^1, \ldots, y^M; \boldsymbol{\mu}, \Sigma), \tag{13}$$

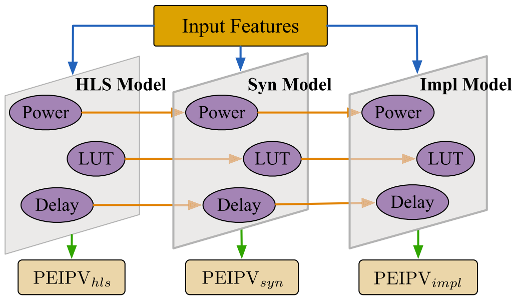


Fig. 10. The combined models, with three stages (fidelities) and three objectives. The orange lines represent the non-linear relationships. The blue lines represent the inputs. Each fidelity has an acquisition function PEIPV.

where $\boldsymbol{\mu}$ is the mean vector with length M and each element μ_i in it is the mean value of objective f^i. The covariance matrix Σ is non-diagonal. Specifically, definition of the covariance value is

$$\Sigma_{i,j} = \mathrm{Cov}(f^i(\boldsymbol{x}),\ f^j(\boldsymbol{x}')) = \mathrm{K}_{i,j} k_C(\boldsymbol{x}, \boldsymbol{x}'), \tag{14}$$

where $\mathrm{K}_{i,j}$ is the similarity between objectives i and j and can be obtained by maximizing likelihood estimation. k_C is a covariance function over $\mathcal{X}$ and is defined as **automatic relevance determination** (**ARD**) Matérn 5/2 kernel to avoid over-smoothness [38].

4.3 Combined Model

Our method has two novel modeling techniques, one for modeling the multiple correlated objectives and one for modeling the three fidelities (i.e., three FPGA design stages). At each fidelity, all of the objectives construct a correlated multi-objective model. Its expected improvement function of Pareto hyper-volume is denoted as $\mathrm{EIPV}_i(\boldsymbol{x}_{t+1}|\mathcal{D})$, with $i \in \{hls, syn, impl\}$. In the real FPGA design flow, obtaining results in different stages costs different running times. To characterize the different costs, an additional penalty term ρ_i is applied to augment $\mathrm{EIPV}_i(\boldsymbol{x}_{t+1})$ as the penalized EIPV, termed as $\mathrm{PEIPV}_i(\boldsymbol{x}_{t+1}|\mathcal{D})$:

$$\begin{aligned} \mathrm{PEIPV}_i(\boldsymbol{x}_{t+1}|\mathcal{D}) &= \rho_i \cdot \mathrm{EIPV}_i(\boldsymbol{x}_{t+1}|\mathcal{D}), \\ \rho_i &= \frac{T_{impl}}{T_i}, i \in \{hls, syn, impl\}, \end{aligned} \tag{15}$$

where T_i is the time of running the FPGA design tool from scratch to stage i. Finally, PEIPV functions in Equation (15) are used as the acquisition functions in the Bayesian optimization framework. For the designs that violate the design rules, no valid reports are returned from the FPGA tool. Their simulation performance is set to be $10\times$ worse than the current worst-case, to punish the illegal designs and teach the models. Figure 10 visualizes the structures of the combined models.

4.4 Optimization Based on Deep Kernel Functions

The properties of the distributions over functions induced by a GP are controlled by the kernel function, and the covariance matrices implicitly depend on the hyper-parameters in the kernel functions [10, 18, 48, 49, 53]. From this perspective, learning better kernel functions are of vital importance to the performance of the GP models. In the recent fewer years, deep neural networks have been shown to have powerful mechanisms to create adaptive functions to discover

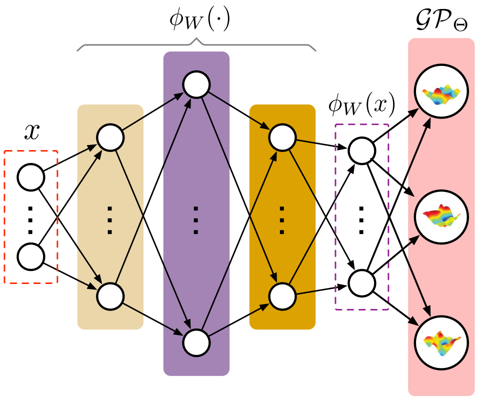


Fig. 11. Our method combines the GP model GP_Θ and the deep kernel function ϕ_W. The original input feature is $\boldsymbol{x}$. And the learned novel feature representation is $\phi_W(\boldsymbol{x})$.

meaningful representations of input data. Therefore, we propose to use the deep neural network as the deep kernel function in the GP models.

As mentioned above, the covariance value in the covariance matrix Σ is defined as Equation (14), where k_C is defined as ARD Matérn 5/2 kernel. ARD Matérn 5/2 kernel [32] takes the form

$$\begin{aligned} k_C(\boldsymbol{x}, \boldsymbol{x}') &= \frac{2^{1-\nu}}{\Gamma(\nu)} \left(\sqrt{2\nu} d\right) K_\nu \left(\sqrt{2\nu} d\right), \\ d &= (\boldsymbol{x} - \boldsymbol{x}')^\top \theta^{-1} (\boldsymbol{x} - \boldsymbol{x}'), \end{aligned} \tag{16}$$

where Γ is the gamma function, ν is the smoothness parameter ($\nu = 5/2$), K_ν is a modified Bessel function of the second kind, and θ is the parameters to be learned. Despite the complicated form, it can be regarded as the inner product of the input feature vectors. Since the covariance only depends on the distances between inputs, it is stationary. In different applications, it is hard to give a general and uniform representation for the different feature vectors to characterize the complicated relationships between the design configurations.

To improve the characterization ability of the kernel function, we propose to use deep neural networks to enhance the kernel function, termed as deep kernel function, as shown in Equation (17).

$$k_C'(\boldsymbol{x}, \boldsymbol{x}') = k_C(\phi_W(\boldsymbol{x}), \phi_W(\boldsymbol{x}') | \theta, W), \tag{17}$$

where $\phi_W(\cdot)$ represents the neural network and W represents the parameters in the network. Implicitly, that is equivalent to learn a novel distance metric to enhance the distance d in Equation (16), i.e.,

$$d' = (\phi_W(\boldsymbol{x}) - \phi_W(\boldsymbol{x}'))^\top \theta^{-1} (\phi_W(\boldsymbol{x}) - \phi_W(\boldsymbol{x}')). \tag{18}$$

The structure of the model with a deep kernel function is shown in Figure 11. The learned feature of the input configuration is $\phi_W(\boldsymbol{x})$. Then $\phi_W(\boldsymbol{x})$ is used as the inputs to the GP models. The structure of our deep kernel function is described in detail in the experiments.

The weight W of the neural network is a part of the parameters in our method. For brevity, denote the multi-objective GP model as GP_Θ with parameters Θ. Parameters Θ and W are optimized jointly, by maximizing the log marginal likelihood $\mathcal{L}$ of the Gaussian process model. According to the chain rule of the gradients, Θ and W can be updated according to

$$\begin{aligned}\frac{\partial \mathcal{L}}{\partial \Theta} &= \frac{\partial \mathcal{L}}{\partial \mathrm{K}_C}\frac{\partial \mathrm{K}_C}{\partial \Theta},\\ \frac{\partial \mathcal{L}}{\partial W} &= \frac{\partial \mathcal{L}}{\partial \mathrm{K}_C}\frac{\partial \mathrm{K}_C}{\partial \phi_W(\boldsymbol{x})}\frac{\partial \phi_W(\boldsymbol{x})}{\partial W},\end{aligned} \tag{19}$$

where K_C denotes the kernel functions, which contain $\mathrm{K}_{i,j}$ in Equation (14) and k'_C in Equation (17), $\frac{\partial \mathrm{K}_C}{\partial \Theta}$ represents the derivatives of the kernel with respect to the kernel parameters. $\frac{\partial \mathrm{K}_C}{\partial \phi_W(\boldsymbol{x})}$ represents the derivatives of the deep kernel with respect to the neural network ϕ_W, while Θ is fixed. With this training method, all of the parameters are trained jointly according to a unified supervised objective, as part of the Gaussian process framework, without requiring approximate Bayesian inference. During training, the gradients are computed via backpropagation.

5 THE OVERALL OPTIMIZATION FLOW

The Bayesian optimization method is adopted as the algorithm skeleton to explore the Pareto-optimal directive configurations, with the GP models as the surrogate models, and PEIPV functions as the acquisition functions.

Using data from later stages contributes to a more accurate surrogate model, at the cost of more simulation workloads to obtain the performance values. If the results at the early FPGA design stage are good enough, there is no need to run the FPGA design tool to the later stages. In each optimization step of the BO algorithm, for the surrogate model of each stage, we need to consider the quality of the selected point and its acquisition value, so as to determine whether it is necessary to optimize models of the later stages. For example, in one BO step, PEIPV_{syn} is the best compared with PEIPV_{hls} and PEIPV_{impl}, at configuration $\boldsymbol{x}_{syn}$. We will then run the FPGA design tool with $\boldsymbol{x}_{syn}$ as the directive configuration input, to get the real performance values at *hls* and *syn* stages. Finally, we will update the surrogate models of *hls* and *syn* stages according to these performance values. If PEIPV_{impl} is the best, we will run the FPGA design tool to the final *impl* stage, and update the models of *hls*, *syn*, and *impl* stages.

The overall optimization flow is detailed in Algorithm 2 and Figure 12. Firstly, we define and prune the design space according to the tree-based method described in Algorithm 1. Denote the generated design space as $\mathcal{X}$. Secondly, we randomly sample some configurations from the design space for initialization. The configurations for the higher fidelities (later FPGA stages) are subsets of the lower fidelities (earlier FPGA stages), i.e., $\mathcal{X}_{impl} \subseteq \mathcal{X}_{syn} \subseteq \mathcal{X}_{hls} \subseteq \mathcal{X}$. These configurations are then fed into the FPGA design tool to get real performance values $\mathcal{Y}_i$, with $i \in \{hls, syn, impl\}$. For each stage, we initialize a surrogate model DKLGP_i (i.e., GP model with deep kernel learning function) and an acquisition function PEIPV_i. In each optimization time step, for each stage i, we will select a configuration $\hat{\boldsymbol{x}}_i \in \mathcal{X}$ which maximizes the expected improvement PEIPV_i. $\hat{\boldsymbol{x}}_i$ is regarded as the candidate Pareto configuration of this stage. Then a node-stage pair $(\boldsymbol{x}^*, h)$, which achieves the highest expected improvement is selected from the three $\hat{\boldsymbol{x}}_i$ configurations. Here h denotes the stage index. $\boldsymbol{x}^*$ is our final choice of Pareto point in current optimization step. A toy example on the surrogate models and PEIPV functions is shown in Figure 13. We will then pass the code together with configuration $\boldsymbol{x}^*$ into the FPGA design tool, run the tool up to stage h to get the performance values (i.e., $\boldsymbol{y}_i$, with $i = hls, \ldots, h$). Record the configuration and performance values, i.e., $\mathcal{X}_i \leftarrow \mathcal{X}_i \cup \{\boldsymbol{x}^*\}$, and $\mathcal{Y}_i \leftarrow \mathcal{Y}_i \cup \{\boldsymbol{y}_i\}$, and update all of the corresponding surrogate models and PEIPV functions. Then we will start the next BO searching step. The final Pareto designs $\{\mathcal{X}^*, \mathcal{Y}^*\}$ found by our optimization method are computed from $\{\mathcal{X}_{impl}, \mathcal{Y}_{impl}\}$.[2]

[2]Note that different from the optimization process, given a set with the known objective values, the Pareto set of this set is deterministic and can be computed easily.

ALGORITHM 2: The Optimization Flow Based on Bayesian Optimization Method

```
1:  Inputs: High-level programming language source code, optimization steps $N_{iter}$, early-
    stopping step $S_E$;
2:  Outputs: Pareto configuration set $\mathcal{X}^*$ and objective value set $\mathcal{Y}^*$, initially $\mathcal{X}^* \leftarrow \emptyset$, $\mathcal{Y}^* \leftarrow \emptyset$;
3:  Enumerate the design space and run tree-based pruning method, to get pruned design space
    $\mathcal{X}$;                                                          ▹ Section 3.1
4:  Randomly sample initial sets $\mathcal{X}_{impl} \subseteq \mathcal{X}_{syn} \subseteq \mathcal{X}_{hls} \subseteq \mathcal{X}$;
5:  Run FPGA tool to get the performance values $\mathcal{Y}_i$ for $\mathcal{X}_i$, with $i \in \{hls, syn, impl\}$;
6:  Initialize a surrogate model DKLGP$_i$ and an acquisition function PEIPV$_i$ for each stage $i$ ac-
    cording to $\{\mathcal{X}_i, \mathcal{Y}_i\}$, with $i \in \{hls, syn, impl\}$;   ▹ DKLGP$_i$ is our method with GPs and deep
    kernel learning functions
7:  for $t \leftarrow 1$ to $N_{iter}$ do
8:      for all stage $i \in \{hls, syn, impl\}$ do
9:          Update DKLGP$_i$ and PEIPV$_i$ according to $\{\mathcal{X}_i, \mathcal{Y}_i\}$;
10:         $\hat{\boldsymbol{x}}_i \leftarrow \arg\max_{\boldsymbol{x} \in \mathcal{X}}$ PEIPV$_i(\boldsymbol{x})$;   ▹ Select the candidate Pareto configuration from $\mathcal{X}$
11:     end for
12:     $(\boldsymbol{x}^*, h) \leftarrow \arg\max_{(\hat{\boldsymbol{x}}_i, i)}$ PEIPV$_i(\boldsymbol{x})$, with $i \in \{hls, syn, impl\}$;   ▹ Determine the Pareto
    configuration
13:     Run FPGA tool with $\boldsymbol{x}^*$ up to stage $h$, to get $\boldsymbol{y}_i$, with $i \in \{hls, \dots, h\}$;
14:     $\mathcal{X}_i \leftarrow \mathcal{X}_i \cup \boldsymbol{x}^*$, $\mathcal{Y}_i \leftarrow \mathcal{Y}_i \cup \boldsymbol{y}_i$, with $i \in \{hls, \dots, h\}$;
15:     $\mathcal{X} \leftarrow \mathcal{X} \setminus \boldsymbol{x}^*$;                ▹ Remove $\boldsymbol{x}^*$ from the design space
16:     Compute current hypervolume $hv_t$ of $\{\mathcal{X}_{impl}, \mathcal{Y}_{impl}\}$;
17:     if $hv_t$ has no improvements in the continuous $S_E$ steps then   ▹ Early-stopping condition
18:         break;                        ▹ Converge and exit the optimization process
19:     end if
20: end for
21: Select Pareto configurations $\{\mathcal{X}^*, \mathcal{Y}^*\}$ from $\{\mathcal{X}_{impl}, \mathcal{Y}_{impl}\}$;   ▹ Obtain the final results from the
    exploration record
22: return Pareto configurations $\{\mathcal{X}^*, \mathcal{Y}^*\}$.
```

Note that in our framework, no additional model training dataset is required. Some configurations are sampled from the design space to initialize the models during the model initialization, i.e., train the model from scratch. These initial configurations with their performance values $\{\mathcal{X}_i, \mathcal{Y}_i\}, i \in \{hls, syn, impl\}$ are the initial training set. Then, new configurations are selected from the design space in the iterative optimization process, according to the expected improvements discussed above. These newly sampled configurations are passed to the FPGA design tool to get actual performance values and extend the training set to tune the model further. The deep kernel modules and GP modules are trained jointly according to Equation (19). The process is repeatedly performed several times until convergence.

Compared with our previous work CGP [40], considering that the shallow structures of Gaussian process models limit the ability to extract information from the input configurations, we combine the deep kernel learning functions with GP models in DCGP, as the enhancement of the kernel functions to learn better feature representations flexibly. Based on this combination, the proposed DCGP is expected to further improve significantly the performance.

Our proposed method can be easily used to explore large design spaces. If the design space is large, our proposed pruning method can effectively prune the design space since some directives are conflicting and some are obviously non-optimal (discussed in Section 3.1). Moreover, the

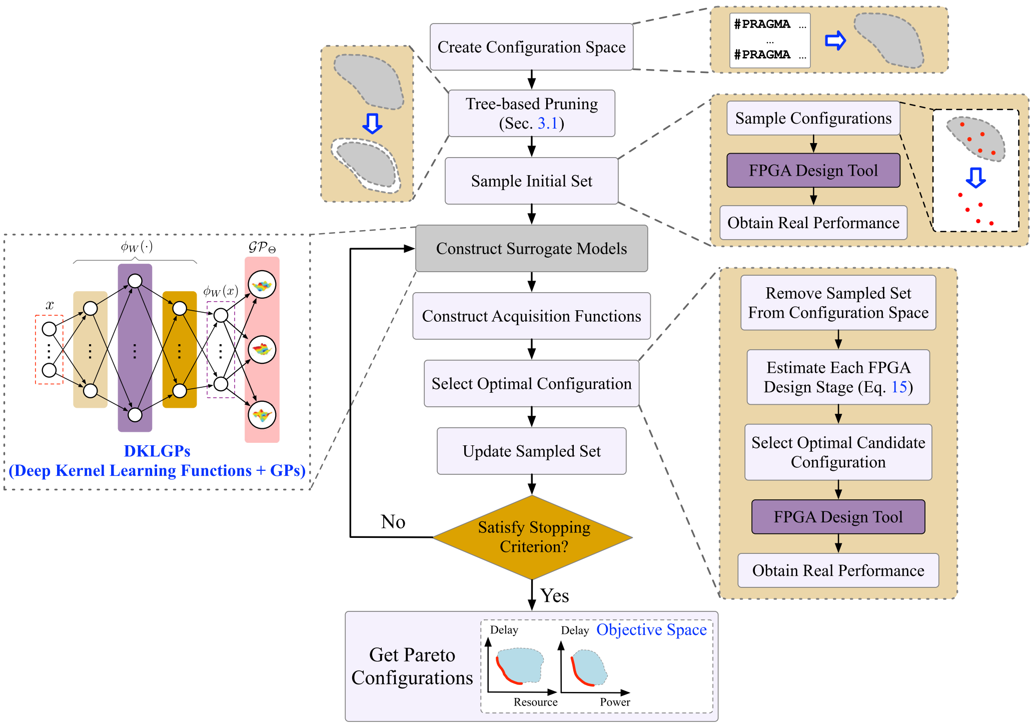

Create Configuration Space
#PRAGMA ...
#PRAGMA ...
Tree-based Pruning (Sec. 3.1)
Sample Initial Set
Sample Configurations
FPGA Design Tool
Obtain Real Performance
Construct Surrogate Models
$\phi_W(\cdot)$
$\mathcal{GP}_\Theta$
x
$\phi_W(x)$
DKLGPs (Deep Kernel Learning Functions + GPs)
Construct Acquisition Functions
Select Optimal Configuration
Update Sampled Set
Satisfy Stopping Criterion?
No
Yes
Remove Sampled Set From Configuration Space
Estimate Each FPGA Design Stage (Eq. 15)
Select Optimal Candidate Configuration
FPGA Design Tool
Obtain Real Performance
Get Pareto Configurations
Objective Space
Delay
Resource
Delay
Power

Fig. 12. Detailed overall optimization flow.

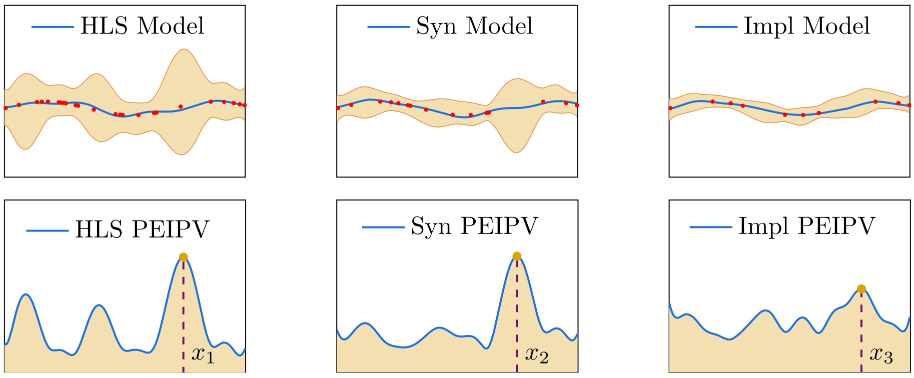


Fig. 13. A toy example to explain the models of the 3 stages (as shown in Figure 3) and their corresponding acquisition functions. Red nodes are the sampled configurations. Models of the lower stages have wider error ranges (light yellow fillers) since their fidelities are lower. Each stage selects a configuration with the highest expected improvement, i.e., x_1, x_2, and x_3. x_1 has the higher expected improvement compared with x_2 and x_3. Therefore, in this optimization step, the HLS stage is selected and x_1 is sampled.

sampling-based method can handle the design space efficiently. The design space is specified by the users via configuration files, while also considering the characteristics of FPGA designs. For example, the sizes of memory blocks are the power of two. Furthermore, considering the required computation workloads in the applications, the numbers of candidate configurations of directives are limited. If in some circumstances the design space is extremely large, we can use sampling-based methods [50, 51] to help solve the problem.

6 EXPERIMENTAL RESULTS

In our experiments, the initial design space is defined by specifying all of the possible locations of directives and their factors in YAML files. We parse the YAML files and convert the directives to feature vectors and HLS TCL files. The target FPGA board is Xilinx Virtex-7 VC707. The FPGA design tool is Xilinx Vivado 2018.2. Our correlated GP version "CGP" [40] is implemented based on [4] and [37]. The deep version "DCGP" is implemented based on BoTorch [1] and GPyTorch [12]. The optimal results in the tables are in bold.

6.1 Objective Selection

Power, performance, and area (**PPA**) are three popular metrics. To measure the system performance, we choose to use delay (task time length), i.e., the product of latency and clock period. Latency reflects how many clock cycles are needed to finish one task. The clock period is the time length of each cycle, which reflects the congestion information of the designs and is a key design indicator for some applications. We use the utilization of the **look-up table** (**LUT**) as the area (i.e., resource consumption) metric. LUT can be used to implement the control logic and simple computations. For tasks requiring high parallelism designs, the LUT utilization is usually the key metric. Other resource metrics (RAMs, DSPs, FFs) can also be easily integrated into the multiple objectives in the same manner. Power as a metric is directly used in this article. Compared to works that consider only one or two metrics or linear combinations of these metrics, our work is more practical and challenging.

6.2 Benchmarks and Methods

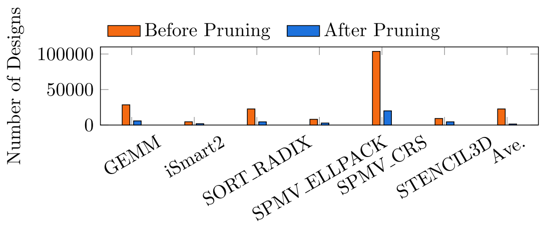
We conduct experiments on six benchmarks. Five are from the open-source FPGA application benchmark MachSuite [33], i.e., `GEMM`, `SORT_RADIX`, `SPMV_ELLPACK`, `SPMV_CRS`, and `STENCIL3D`. Another benchmark is `iSmart2` [17], an object detection deep neural network model deployed on FPGA. `GEMM` is the general matrix multiplication which is widely used to implement the convolutional operators, fully connected operations, and so on. Both `SPMV_ELLPACK` and `SPMV_CRS` are for the sparse matrix-vector multiplication while their storage formats are different (`ELLPACK` format and `CRS` format). These two sparse computations are needed by the sparse neural networks. `iSmart2` stacks 12 convolutional modules, including group convolutions, point-wise convolutions, ReLU, pooling layers, and so on. `Stencil3D` is the stencil operation used in numerical analysis. We consider the unrolling and pipelining for the loops and partitioning for the arrays. BRAM core is used as the storage resource. DSP48 and Mul_LUT are used as the computation resource cores. Each benchmark contains a large number of possible configurations. The average running times of the FPGA tool for these designs are shown in Figure 14(a). The sizes of the design spaces before and after the tree-based pruning method are plotted in Figure 14(b).

In the deep kernel function of DCGP, there are four fully connected layers, with output dimensions 1,000, 500, 50, and 6. Each of the first three fully connected layers is appended with a ReLU layer. The outputs of the deep kernel functions are the inputs of the GP models. The input features are enlarged into a high-dimension space (i.e., 1,000, and 500) by the deep model to learn more information. The dimension is very large compared with the original feature configurations, and that is why this is called the deep method. Then the features are embedded to learn key information with smaller dimensions (i.e., 50, and 6).

Four popular and representative methods are compared with our methods. [25], shorted as FPL18, is also based on Bayesian methods and the Gaussian process. The authors build linear

Benchmark	HLS (s)	Syn (s)	Impl (s)
GEMM	137.75	1379.67	1744.61
iSmart2	2206.06	2895.15	4022.38
SORT_RADIX	166.66	917.01	1206.94
SPMV_ELLPACK	365.91	1528.81	1034.79
SPMV_CRS	1333.95	3614.51	1503.30
STENCIL3D	369.10	2082.25	969.29
Average	763.24	2069.57	1746.89

(a)



(b)

Fig. 14. (a) Average running time of the FPGA tool for each design; (b) Sizes of the design spaces before and after the tree-based pruning.

multi-fidelity and independent multi-objective models. [22], abbreviated as DAC19, defines several regression models to guide the FPGA HLS designs with existing ASIC designs. Although the starting points are different, their methods are transferable. *Post-HLS* reports in our problem can be regarded as the ASIC implementations, to predict the *post-Implementation* reports. RTED proposed by [21] is also used in DAC19 [22] to select better and representative initialization configurations. **Artificial neural network** (**ANN**) and **Boosting tree** (**BT**) methods have been used in [9, 41, 56] to guide the back-end designs and achieve good performances. For these regression algorithms, some configurations are randomly sampled from the design space to initialize these models. We use the *post-Implementation* reports as the regression targets. For each objective, we build one model. After all of the models are trained, the whole design space is fed into these models to predict the Pareto points. For these learned Pareto points, we run the Xilinx Vivado design flow to get their real reports. Note that these models are only used to learn the relative numerical relationships to determine the Pareto points. Besides, the various design objectives have performance values that vary greatly in order of magnitude. To guarantee the stability and robustness of the trained models, the performance values are divided by estimated values to scale different target values to a suitable range. For fairness, all of these algorithms use the same feature encodings and design spaces as our method.

Different design objectives have distinctive ranges of performance values, which would mislead the models significantly. For example, the latencies are several seconds while the consumptions of LUTs are hundreds of thousands. The objective values should be normalized during optimizations to avoid inappropriate data shifts. Considering that the maximum power and hardware resources are specified for a known FPGA platform, in practice, we can use these specifications to help adjust the data magnitude. For delay, we estimate a scale factor according to the delays obtained in the initialization stages ($2\times$ of the maximum in the initialization stages). Then the delays are divided by the scale factor before feeding into the model.

6.3 Experimental Settings

For our methods and FPL18 [25], we run 10 tests on each benchmark and the results reported in this article are the averages. For each benchmark, 8 configurations are randomly sampled to initialize the models. The maximum optimization step is 40 and the early stopping factor is 5. There exists a trade-off between the optimization costs and the quality of results. The experimental results show that our methods converge after 30 steps. Therefore, we choose to use 40 steps.

For ANN, we design a model with 2 hidden layers. We train the model with $\{500, 1,000, \ldots, 5,000\}$ times. For the Boosting method used in [9, 41, 56], we run a group of experiments, with tree depth from 1 to 6, and learning rate in $\{0.1, 0.2, 0.3, 0.4, 0.5\}$. In DAC19

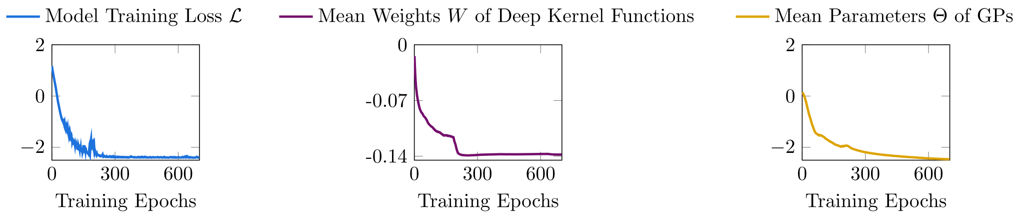


Fig. 15. The training loss of our method, and the mean parameters (W and Θ) of the deep kernel functions and the GP models. The results show the great convergence of the training loss. The mean parameters W and Θ follow the same convergence trend which validates the effectiveness of our training method.

[22], different numbers of initial sets are sampled to build the models. Therefore, the number of initial sets is also considered as a hyper-parameter, i.e., $\{3, 4, \ldots, 11\}$. In experiments of ANN, Boosting, and DAC19, for each benchmark, the number of initialization configurations is 48. It is worth mentioning that some optimization techniques require initialization and iterative optimization (e.g., ours and FPL18), while some methods have only initialization steps (e.g., ANN). In practical applications, with or without initializations and iterative optimizations is not an issue since the target is to find the optimal solutions. They both run the FPGA design tools to get the actual performance values. For each benchmark, we optimize the configurations from scratch with no prior data. Therefore, the fair comparisons should consider the overall costs of the initializations and the iterative optimizations for different optimization techniques. This kind of cost measures the overall overhead for the users to achieve final optimization results. The overall costs are compared in Section 6.6. Although the maximum optimization step is 40, the optimization step is usually fewer than 40 because of the early stopping mechanism (i.e., convergence).

Two metrics are used to measure the performance: **average distance to reference set** (**ADRS**) and overall running time. ADRS computes the distance between the learned Pareto set and the real Pareto set [22].

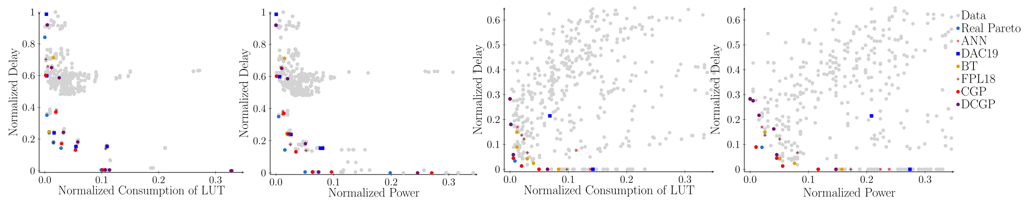
$$\mathrm{ADRS}(\Gamma, \Omega) = \frac{1}{|\Gamma|} \sum_{\gamma \in \Gamma} \min_{\omega \in \Omega} f(\gamma, \omega), \tag{20}$$

where Ω is the learned Pareto set, Γ is the real Pareto set, $f(\gamma, \omega)$ is the distance between two points, $\gamma \in \Gamma$ and $\omega \in \Omega$, $|\Gamma|$ is the number of points in Γ. Overall running time is the total time needed to get all results, including initialization and iterative optimization. To validate the effectiveness of our method, all the configurations in the design space are run to obtain the whole objective space, though consuming lots of time, huge amounts of computation, and storage resources.

6.4 Results and Analysis

As mentioned above, all of the parameters are trained jointly with a unified supervised objective based on the chain rule, as shown in Equation (19). An example of the training loss and parameters of our deep kernel function and GP model is shown in Figure 15. The results show that the parameters of the deep kernel function and the GP model have good convergence trends, which validates the performance of our joint training method.

Two examples are plotted in Figure 16 to show the learned Pareto points. For easy visualizations, three objectives are plotted in two figures. The results demonstrate that our learned Pareto points are much more closer to the real Pareto points. All of the statistical results are listed in Table 1, while expressed as ratios to the results of ANN.



(a) `GEMM LUT-Delay` (b) `GEMM Power-Delay` (c) `SPMV_ELLPACK LUT-Delay` (d) `SPMV_ELLPACK Power-Delay`

Fig. 16. Learned Pareto designs of GEMM and SPMV_ELLPACK in the objective spaces.

Table 1. Normalized Experimental Results

Benchmark	Normalized ADRS						Normalized Standard Deviation of ADRS					
	FPL18	ANN	BT	DAC19	CGP	**DCGP**	FPL18	ANN	BT	DAC19	CGP	**DCGP**
`GEMM`	0.50	1.00	0.65	1.08	0.27	**0.25**	0.46	1.00	0.37	0.90	**0.12**	0.19
`iSmart2`	0.68	1.00	1.28	1.49	0.65	**0.59**	0.75	1.00	1.10	1.24	**0.20**	0.26
`SORT_RADIX`	0.72	1.00	1.09	0.94	0.64	**0.59**	0.57	1.00	1.72	2.28	0.48	**0.27**
`SPMV_ELLPACK`	0.47	1.00	0.22	1.21	0.19	**0.11**	0.24	1.00	0.06	0.99	0.09	**0.01**
`SPMV_CRS`	0.29	1.00	2.09	1.15	0.22	**0.20**	0.26	1.00	2.09	1.52	**0.03**	0.20
`STENCIL3D`	0.41	1.00	0.40	0.41	0.39	**0.31**	0.57	1.00	**0.00**	0.05	0.03	0.05
Average	0.51	1.00	0.96	1.05	0.39	**0.34**	0.47	1.00	0.89	1.16	0.16	**0.16**

Table 2. Profiling Information of Running Times (Hours)

Benchmark	FPL18			ANN			BT			DAC19			CGP			DCGP		
	Alg.	FPGA	Overall	Alg.	FPGA	Overall	Alg.	FPGA	Overall	Alg.	FPGA	Overall	Alg.	FPGA	Overall	Alg.	FPGA	Overall
`GEMM`	0.46	24.92	25.38	0.05	30.03	30.08	0.21	30.03	30.24	0.09	210.21	210.30	0.54	20.42	20.96	0.63	18.48	**19.11**
`iSmart2`	0.41	127.83	128.24	0.12	145.26	145.38	0.39	145.26	145.65	0.22	1016.82	1017.04	0.56	61.01	61.57	0.58	59.07	**59.65**
`SORT_RADIX`	1.02	15.19	16.21	0.98	32.32	33.30	0.98	32.32	33.30	1.02	226.24	227.26	1.17	10.99	12.16	1.06	10.47	**11.53**
`SPMV_ELLPACK`	0.37	16.28	16.65	0.03	38.77	38.80	0.27	38.77	39.04	0.05	271.39	271.44	0.48	25.20	25.68	0.72	19.25	**19.97**
`SPMV_CRS`	0.48	77.18	77.66	0.02	85.75	85.77	0.22	85.75	85.97	0.06	600.25	600.31	0.58	61.74	62.32	0.73	60.32	**61.05**
`STENCIL3D`	0.54	18.59	19.13	0.07	45.34	45.41	0.43	45.34	45.77	0.12	317.38	317.50	0.63	19.95	20.58	0.42	17.44	**17.86**
Average	0.55	46.67	47.21	0.21	62.91	63.12	0.42	62.91	63.33	0.26	440.38	440.64	0.66	33.22	33.88	0.69	29.39	**30.08**

As shown in Table 1, our methods, CGP [40] and DCGP outperform all of these baselines by a lot. Firstly, compared with FPL18 [25], our methods can achieve much better results on all benchmarks. That is because we consider practical non-linear and correlated relationships in real applications. Secondly, the other three methods are also worse than ours, because they cannot handle complex relationships between multiple fidelities. Thirdly, for benchmarks with complicated code structures, the models without GP models are inferior. For example, the irregular memory accesses of `SORT_RADIX` bring great challenges to ANN, Boosting tree, and DAC19. The results prove that our methods are general enough to handle various applications. Our methods also achieve much better stability according to the standard deviations of ADRSs, as shown in Table 1. Besides, our deep version DCGP outperforms CGP [40] on all of the six benchmarks with respect to the Pareto results with lower ADRS values, and the same average standard derivations. These results effectively prove the performance of the deep method proposed in Section 4.4.

The averages of overall running time are listed in Table 2 to show that our methods can also save time. For fair, the FPGA times are computed according to the average times listed in Figure 14(a) and the number of interactions with the FPGA tool. For DAC19, the size of one training data set equals ANN. But it has $3 \sim 11$ training sets. Therefore the average FPGA running time is $7\times$ (i.e., $(3+11)/2 = 7$) greater than ANN and Boosting tree. DCGP also consumes less time than CGP [40], thanks to the fewer interactions with FPGA tools. Though DCGP is more complicated compared

Table 3. Comparisons of Normalized ADRS with MES

Benchmark	DCGP	MES
GEMM	**0.25**	0.47
iSmart2	**0.59**	0.61
SORT_RADIX	**0.59**	0.66
SPMV_ELLPACK	**0.11**	0.30
SPMV_CRS	**0.20**	0.26
STENCIL3D	**0.31**	0.40
Average	**0.34**	0.45

with CGP, the costs are acceptable with several minutes longer "Alg." times. More analyses are provided in Section 6.6.

In summary, our DCGP outperforms our CGP [40] and the other baselines. Numerically, the DCGP wins CGP 12.8% on average ADRS and up to 42.1% on SPMV_ELLPACK, thanks to the outstanding performance of using deep kernel learning functions and the joint training method.

6.5 Ablation Studies on Acquisition Function

In this section, we compare our acquisition function with **max-value entropy search** (**MES**), another popular acquisition function for multi-objective optimization problems. In literature [3, 15], the objectives $f^1(\boldsymbol{x}), f^2(\boldsymbol{x}), \ldots, f^M(\boldsymbol{x})$ are modeled using M independent GP models with zero mean and *i.i.d.* noise. Researchers propose maximizing the information gains with respect to the Pareto designs learned in the previous optimization steps to overcome the challenges of computing the acquisition function based on input space's entropy. The GP priors are approximated as $\tilde{f}^i$ and sampled from the M independent GP models [3, 15, 16, 44]. We implement MES and embed it into our optimization framework in place of our correlated acquisition function. The experimental results are listed in Table 3, under the same experimental settings as our DCGP. The results show that using MES degrades performance. The results prove the outstanding performance of our framework with correlated multi-objective optimization methods, which are important contributions of this article.

6.6 Ablation Studies on Running Time

The DCGP method accelerates the optimization process significantly compared with baselines because it requires fewer optimization iterations. Though more time is needed to train the model in each optimization iteration, the training workload is tiny and can be finished in minutes. The time costs reduced by interacting fewer with the FPGA design flow are exciting. The details are listed in Table 2. "Alg." denotes the time costs of running the optimization algorithms, and "FPGA" represents the time costs to run the FPGA design tool to get the actual performance. The results show that using DCGP reduces the time costs remarkably compared with the baselines.

7 CONCLUSION

In this article, we solve the problem of FPGA HLS directives design optimization. A tree-based pruning method is proposed to prune the design space. Correlated multi-objective multi-fidelity Gaussian process (CGP) models can handle the strong nonlinearities among the multiple fidelities. To the best of our knowledge, the correlated multi-fidelity model is introduced into the HLS directive optimization domain for the first time and has been proven to be effective. The advanced deep version further improves the qualities of the learned Pareto points with shorter running times, by using deep neural networks to enhance the kernel functions. We hope this article will stimulate new research directions in this domain. The public benchmarks, e.g., GEMM and SPMV, and an objective detection deep neural network iSmart2 are tested. The results prove the outstanding performance of our method.

REFERENCES

[1] Maximilian Balandat, Brian Karrer, Daniel Jiang, Samuel Daulton, Ben Letham, Andrew G. Wilson, and Eytan Bakshy. 2020. BoTorch: A framework for efficient Monte-Carlo Bayesian optimization. In *Proceedings of the Advances in Neural Information Processing Systems* . 21524–21538. Retrieved from https://proceedings.neurips.cc/paper/2020/hash/f5b1b89d98b7286673128a5fb112cb9a-Abstract.html.

[2] Samridhi Bansal, Hsuan Hsiao, Tomasz Czajkowski, and Jason H. Anderson. 2018. High-level synthesis of software-customizable floating-point cores. In *Proceedings of the IEEE/ACM Design, Automation and Test in Eurpoe Conference & Exhibition*. 37–42. DOI: https://doi.org/10.23919/DATE.2018.8341976

[3] Syrine Belakaria, Aryan Deshwal, and Janardhan Rao Doppa. 2019. Max-value entropy search for multi-objective Bayesian optimization. *Advances in Neural Information Processing Systems* 32 (2019), 7825–7835. Retrieved from https://proceedings.neurips.cc/paper/2019/hash/82edc5c9e21035674d481640448049f3-Abstract.html.

[4] Edwin V. Bonilla, Kian M. Chai, and Christopher Williams. 2008. Multi-task Gaussian process prediction. In *Proceedings of the Advances in Neural Information Processing Systems*. 153–160. Retrieved from https://proceedings.neurips.cc/paper/2007/hash/66368270ffd51418ec58bd793f2d9b1b-Abstract.html.

[5] Jason Brownlee. 2020. *Data Preparation for Machine Learning: Data Cleaning, Feature Selection, and Data Transforms in Python*. Machine Learning Mastery.

[6] Tinghuan Chen, Bingqing Lin, Hao Geng, Shiyan Hu, and Bei Yu. 2021. Leveraging spatial correlation for sensor drift calibration in smart building. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 40, 7 (2021), 1273–1286. DOI: https://doi.org/10.1109/TCAD.2020.3015438

[7] Tinghuan Chen, Bingqing Lin, Hao Geng, and Bei Yu. 2019. Sensor drift calibration via spatial correlation model in smart building. In *Proceedings of the ACM/IEEE Design Automation Conference*. 1–6. DOI: https://doi.org/10.1145/3316781.3317909

[8] Tinghuan Chen, Qi Sun, and Bei Yu. 2021. Machine learning in nanometer AMS design-for-reliability (invited paper). In *Proceedings of the IEEE International Conference on ASIC*. 1–4. DOI: https://doi.org/10.1109/ASICON52560.2021.9620496

[9] Steve Dai, Yuan Zhou, Hang Zhang, Ecenur Ustun, Evangeline FY Young, and Zhiru Zhang. 2018. Fast and accurate estimation of quality of results in high-level synthesis with machine learning. In *Proceedings of the IEEE International Symposium on Field-Programmable Custom Computing Machines*. 129–132. DOI: https://doi.org/10.1109/FCCM.2018.00029

[10] Alexander G. de G. Matthews, Jiri Hron, Mark Rowland, Richard E. Turner, and Zoubin Ghahramani. 2018. Gaussian Process Behaviour in Wide Deep Neural Networks. In *International Conference on Learning Representations (ICLR)*. https://arxiv.org/abs/1804.11271.

[11] Lorenzo Ferretti, Giovanni Ansaloni, and Laura Pozzi. 2018. Lattice-traversing design space exploration for high level synthesis. In *Proceedings of the IEEE International Conference on Computer Design*. 210–217. DOI: https://doi.org/10.1109/ICCD.2018.00040

[12] Jacob Gardner, Geoff Pleiss, Kilian Q. Weinberger, David Bindel, and Andrew G. Wilson. 2018. GPyTorch: Blackbox matrix-matrix Gaussian process inference with GPU acceleration. In *Proceedings of the Advances in Neural Information Processing Systems*. 7587–7597. Retrieved from https://proceedings.neurips.cc/paper/2018/hash/27e8e17134dd7083b050476733207ea1-Abstract.html.

[13] Hao Geng, Yuzhe Ma, Qi Xu, Jin Miao, Subhendu Roy, and Bei Yu. 2021. High-speed adder design space exploration via graph neural processes. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*. 1–1. DOI: https://doi.org/10.1109/TCAD.2021.3114262

[14] Cong Hao, Xiaofan Zhang, Yuhong Li, Sitao Huang, Jinjun Xiong, Kyle Rupnow, Wen-mei Hwu, and Deming Chen. 2019. FPGA/DNN Co-Design: An efficient design methodology for IoT intelligence on the edge. In *Proceedings of the ACM/IEEE Design Automation Conference*. 1–6. Retrieved from https://ieeexplore.ieee.org/abstract/document/8807043.

[15] Daniel Hernández-Lobato, Jose Hernandez-Lobato, Amar Shah, and Ryan Adams. 2016. Predictive entropy search for multi-objective Bayesian optimization. In *Proceedings of the International Conference on Machine Learning*. PMLR, 1492–1501. Retrieved from http://proceedings.mlr.press/v48/hernandez-lobatoa16.html.

[16] José Miguel Hernández-Lobato, Matthew W. Hoffman, and Zoubin Ghahramani. 2014. Predictive entropy search for efficient global optimization of black-box functions. In *Proceedings of the Advances in Neural Information Process-*

ing Systems. 918–926. Retrieved from https://papers.nips.cc/paper/2014/hash/069d3bb002acd8d7dd095917f9efe4cb-Abstract.html.

[17] iSmartDNN. Retrieved from https://github.com/onioncc/iSmartDNN. (????).

[18] Mohammad Emtiyaz E. Khan, Alexander Immer, Ehsan Abedi, and Maciej Korzepa. 2019. Approximate inference turns deep networks into Gaussian processes. In *Proceedings of the Advances in Neural Information Processing Systems.* 3094–3104. Retrieved from https://proceedings.neurips.cc/paper/2019/hash/b3bbccd6c008e727785cb81b1aa08ac5-Abstract.html.

[19] Martin Letras, Joachim Falk, and Juergen Teich. 2021. Decision tree-based throughput estimation to accelerate design space exploration for multi-core applications. In *MBMV 2021; 24th Workshop.* 1–11. Retrieved from https://ieeexplore.ieee.org/abstract/document/9399720.

[20] Zipeng Li, Tsung-Yi Ho, Kelvin Yi-Tse Lai, Krishnendu Chakrabarty, Po-Hsien Yu, and Chen-Yi Lee. 2016. High-level synthesis for micro-electrode-dot-array digital microfluidic biochips. In *Proceedings of the ACM/IEEE Design Automation Conference.* 1–6. DOI: https://doi.org/10.1145/2897937.2898028

[21] Hung-Yi Liu and Luca P. Carloni. 2013. On learning-based methods for design-space exploration with high-level synthesis. In *Proceedings of the ACM/IEEE Design Automation Conference.* 1–7. DOI: https://doi.org/10.1145/2463209.2488795

[22] Shuangnan Liu, Francis Lau, and Benjamin Carrion Schafer. 2019. Accelerating FPGA prototyping through predictive model-based HLS design space exploration. In *Proceedings of the ACM/IEEE Design Automation Conference.* 1–6. DOI: https://doi.org/10.1145/3316781.3317754

[23] Siting Liu, Qi Sun, Peiyu Liao, Yibo Lin, and Bei Yu. 2021. Global placement with deep learning-enabled explicit routability optimization. In *Proceedings of the IEEE/ACM Design, Automation and Test in Eurpoe Conference & Exhibition.* 1821–1824. DOI: https://doi.org/10.23919/DATE51398.2021.9473959

[24] Charles Lo and Paul Chow. 2016. Model-based optimization of high-level synthesis directives. In *Proceedings of the IEEE International Conference on Field Programmable Logic and Applications.* 1–10. DOI: https://doi.org/10.1109/FPL.2016.7577358

[25] Charles Lo and Paul Chow. 2018. Multi-fidelity optimization for high-level synthesis directives. In *Proceedings of the IEEE International Conference on Field Programmable Logic and Applications.* 272–277. DOI: https://doi.org/10.1109/FPL.2018.00054

[26] Wenlong Lyu, Fan Yang, Changhao Yan, Dian Zhou, and Xuan Zeng. 2018. Batch Bayesian optimization via multi-objective acquisition ensemble for automated analog circuit design. In *Proceedings of the International Conference on Machine Learning.* 3312–3320. Retrieved from http://proceedings.mlr.press/v80/lyu18a.html.

[27] Anushree Mahapatra and Benjamin Carrion Schafer. 2014. Machine-learning based simulated annealer method for high level synthesis design space exploration. In *Proceedings of the 2014 Electronic System Level Synthesis Conference.* IEEE, 1–6. DOI: https://doi.org/10.1109/ESLsyn.2014.6850383

[28] Sebastian W. Ober, Carl E. Rasmussen, and Mark van der Wilk. 2021. The promises and pitfalls of deep kernel learning. In *Proceedings of the Thirty-Seventh Conference on Uncertainty in Artificial Intelligence (UAI)*, Vol. 161. PMLR, 1206–1216. https://proceedings.mlr.press/v161/ober21a.html.

[29] Kenneth O'Neal, Mitch Liu, Hans Tang, Amin Kalantar, Kennen DeRenard, and Philip Brisk. 2018. HLSPredict: Cross platform performance prediction for FPGA high-level synthesis. In *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design.* 1–8. DOI: https://doi.org/10.1145/3240765.3240816

[30] Adrien Prost-Boucle, Olivier Muller, and Frédéric Rousseau. 2013. A fast and autonomous HLS methodology for hardware accelerator generation under resource constraints. In *Proceedings of the Euromicro Conference on Digital System Design.* IEEE, 201–208. DOI: https://doi.org/10.1109/DSD.2013.30

[31] Joaquin Quinonero-Candela and Carl Edward Rasmussen. 2005. A unifying view of sparse approximate Gaussian process regression. *The Journal of Machine Learning Research* 6, 65 (2005), 1939–1959. Retrieved from https://www.jmlr.org/beta/papers/v6/quinonero-candela05a.html.

[32] Carl Edward Rasmussen and C. Williams. 2006. Gaussian processes for machine learning. The MIT Press, *Cambridge, MA.* DOI: https://doi.org/10.1007/978-3-540-28650-9_4

[33] Brandon Reagen, Robert Adolf, Yakun Sophia Shao, Gu-Yeon Wei, and David Brooks. 2014. Machsuite: Benchmarks for accelerator design and customized architectures. In *Proceedings of the 2014 IEEE International Symposium on Workload Characterization.* 110–119. DOI: https://doi.org/10.1109/IISWC.2014.6983050

[34] Hitesh Sapkota, Yiming Ying, Feng Chen, and Qi Yu. 2021. Distributionally robust optimization for deep kernel multiple instance learning. In *Proceedings of the International Conference on Artificial Intelligence and Statistics.* PMLR, 2188–2196. Retrieved from http://proceedings.mlr.press/v130/sapkota21a.html.

[35] Benjamin Carrion Schafer and Kazutoshi Wakabayashi. 2009. Design space exploration acceleration through operation clustering. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 29, 1 (2009), 153–157. DOI: https://doi.org/10.1109/TCAD.2009.2035579

[36] Benjamin Carrion Schafer and Kazutoshi Wakabayashi. 2012. Divide and conquer high-level synthesis design space exploration. *ACM Transactions on Design Automation of Electronic Systems* 17, 3 (2012), 1–19. DOI: https://doi.org/10.1145/2209291.2209302

[37] Amar Shah and Zoubin Ghahramani. 2016. Pareto frontier learning with expensive correlated objectives. In *Proceedings of the International Conference on Machine Learning*. 1919–1927. Retrieved from http://proceedings.mlr.press/v48/shahc16.html.

[38] Jasper Snoek, Hugo Larochelle, and Ryan P. Adams. 2012. Practical Bayesian optimization of machine learning algorithms. In *Proceedings of the Advances in Neural Information Processing Systems*. 2951–2959. Retrieved from https://proceedings.neurips.cc/paper/2012/hash/05311655a15b75fab86956663e1819cd-Abstract.html.

[39] Qi Sun, Tinghuan Chen, Jin Miao, and Bei Yu. 2019. Power-driven DNN dataflow optimization on FPGA. In *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design*. 1–7. DOI: https://doi.org/10.1109/ICCAD45719.2019.8942085

[40] Qi Sun, Tinghuan Chen, Liu Siting, Jin Miao, Jianli Chen, Hao Yu, and Bei Yu. 2021. Correlated multi-objective multi-fidelity optimization for HLS directives design. In *Proceedings of the IEEE/ACM Design, Automation and Test in Eurpoe Conference & Exhibition*. 46–51. DOI: https://doi.org/10.23919/DATE51398.2021.9474241

[41] Ecenur Ustun, Shaojie Xiang, Jinny Gui, Cunxi Yu, and Zhiru Zhang. 2019. Lamda: Learning-assisted multi-stage autotuning for FPGA design closure. In *Proceedings of the IEEE International Symposium on Field-Programmable Custom Computing Machines*. 74–77. DOI: https://doi.org/10.1109/FCCM.2019.00020

[42] Vivado Design Suite User Guide: High-Level Synthesis. Retrieved from https://www.xilinx.com/support/documentation/sw_manuals/xilinx2018_3/ug902-vivado-high-level-synthesis.pdf. (????).

[43] Michael Volpp, Lukas P. Fröhlich, Kirsten Fischer, Andreas Doerr, Stefan Falkner, Frank Hutter, and Christian Daniel. 2020. Meta-learning acquisition functions for transfer learning in Bayesian optimization. In *Proceedings of the International Conference on Learning Representations*. Retrieved from https://openreview.net/forum?id=ryeYpJSKwr.

[44] Zi Wang and Stefanie Jegelka. 2017. Max-value entropy search for efficient Bayesian optimization. In *Proceedings of the International Conference on Machine Learning*. PMLR, 3627–3635. Retrieved from http://proceedings.mlr.press/v70/wang17e.html.

[45] Xuechao Wei, Yun Liang, and Jason Cong. 2019. Overcoming data transfer bottlenecks in FPGA-based DNN accelerators via layer conscious memory management. In *Proceedings of the ACM/IEEE Design Automation Conference*. 125–1. DOI: https://doi.org/10.1145/3316781.3317875

[46] Xuechao Wei, Cody Hao Yu, Peng Zhang, Youxiang Chen, Yuxin Wang, Han Hu, Yun Liang, and Jason Cong. 2017. Automated systolic array architecture synthesis for high throughput CNN inference on FPGAs. In *Proceedings of the ACM/IEEE Design Automation Conference*. 1–6. DOI: https://doi.org/10.1145/3061639.3062207

[47] Andrew Wilson and Hannes Nickisch. 2015. Kernel interpolation for scalable structured Gaussian processes (KISS-GP). In *Proceedings of the International Conference on Machine Learning*. PMLR, 1775–1784. Retrieved from http://proceedings.mlr.press/v37/wilson15.html.

[48] Andrew Gordon Wilson, Zhiting Hu, Ruslan Salakhutdinov, and Eric P. Xing. 2016. Deep kernel learning. In *Proceedings of the International Conference on Artificial Intelligence and Statistics*. 370–378. Retrieved from http://proceedings.mlr.press/v51/wilson16.html.

[49] Andrew G. Wilson, Zhiting Hu, Russ R. Salakhutdinov, and Eric P. Xing. 2016. Stochastic variational deep kernel learning. In *Proceedings of the Advances in Neural Information Processing Systems*. 2586–2594. Retrieved from https://proceedings.neurips.cc/paper/2016/hash/bcc0d400288793e8bdcd7c19a8ac0c2b-Abstract.html.

[50] James T. Wilson, Frank Hutter, and Marc Peter Deisenroth. 2018. Maximizing acquisition functions for Bayesian optimization. In *Proceedings of the Advances in Neural Information Processing Systems*, Vol. 31. 9884–9895. Retrieved from https://papers.nips.cc/paper/2018/hash/498f2c21688f6451d9f5fd09d53edda7-Abstract.html.

[51] James T. Wilson, Riccardo Moriconi, Frank Hutter, and Marc Peter Deisenroth. 2017. The reparameterization trick for acquisition functions. In *Workshop on Bayesian Optimization of Conference on Neural Information Processing Systems (NeurIPS)*. https://arxiv.org/abs/1712.00424

[52] Hanchen Ye, Xiaofan Zhang, Zhize Huang, Gengsheng Chen, and Deming Chen. 2020. HybridDNN: A framework for high-performance hybrid DNN accelerator design and implementation. In *Proceedings of the ACM/IEEE Design Automation Conference*. IEEE, 1–6. DOI: https://doi.org/10.1109/DAC18072.2020.9218684

[53] Zixuan Yin, Warren Gross, and Brett H. Meyer. 2020. Probabilistic sequential multi-objective optimization of convolutional neural networks. In *Proceedings of the IEEE/ACM Design, Automation and Test in Eurpoe Conference & Exhibition*. IEEE, 1055–1060. DOI: https://doi.org/10.23919/DATE48585.2020.9116535

[54] Jieru Zhao, Liang Feng, Sharad Sinha, Wei Zhang, Yun Liang, and Bingsheng He. 2017. COMBA: A comprehensive model-based analysis framework for high level synthesis of real applications. In *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design*. 430–437. DOI: https://doi.org/10.1109/ICCAD.2017.8203809

[55] Jieru Zhao, Liang Feng, Sharad Sinha, Wei Zhang, Yun Liang, and Bingsheng He. 2019. Performance modeling and directives optimization for high level synthesis on FPGA. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems.* 39, 7 (2021), 1428–1441. DOI: https://doi.org/10.1109/TCAD.2019.2912916

[56] Jieru Zhao, Tingyuan Liang, Sharad Sinha, and Wei Zhang. 2019. Machine learning based routing congestion prediction in FPGA high-level synthesis. In *Proceedings of the IEEE/ACM Design, Automation and Test in Eurpoe Conference & Exhibition.* 1130–1135. DOI: https://doi.org/10.23919/DATE.2019.8714724

[57] Guanwen Zhong, Alok Prakash, Yun Liang, Tulika Mitra, and Smail Niar. 2016. Lin-analyzer: A high-level performance analysis tool for FPGA-based accelerators. In *Proceedings of the ACM/IEEE Design Automation Conference.* 1–6. DOI: https://doi.org/10.1145/2897937.2898040

[58] Wei Zuo, Warren Kemmerer, Jong Bin Lim, Louis-Noël Pouchet, Andrey Ayupov, Taemin Kim, Kyungtae Han, and Deming Chen. 2015. A polyhedral-based systemc modeling and generation framework for effective low-power design space exploration. In *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design.* IEEE, 357–364. DOI: https://doi.org/10.1109/ICCAD.2015.7372592

Received July 2021; revised October 2021; accepted December 2021