Adaptive Threshold Non-Pareto Elimination: Re-thinking Machine Learning for System Level Design Space Exploration on FPGAs

Pingfan Meng, Alric Althoff, Quentin Gautier, and Ryan Kastner
Department of Computer Science and Engineering, University of California, San Diego, USA
Email: {pmeng, aalthoff, qkgautier, kastner}@cs.ucsd.edu

***Abstract*—One major bottleneck of the system level OpenCL-to-FPGA design tools is their extremely time consuming synthesis process (including place and route). The design space for a typical OpenCL application contains thousands of possible designs even when considering a small number of design space parameters. It costs months of compute time to synthesize all these possible designs into end-to-end FPGA implementations. Thus, the brute force design space exploration (DSE) is impractical for these design tools.**

Machine learning is one solution that identifies the valuable Pareto designs by sampling only a small portion of the entire design space. However, most of the existing machine learning frameworks focus on improving the design objective regression accuracy, which is not necessarily suitable for the FPGA DSE task. To address this issue, we propose a novel strategy - Adaptive Threshold Non-Pareto Elimination (ATNE). Instead of focusing on regression accuracy improvement, ATNE focuses on understanding and estimating the inaccuracy. ATNE provides a Pareto identification threshold that adapts to the estimated inaccuracy of the regressor. This adaptive threshold results in a more efficient DSE. For the same prediction quality, ATNE reduces the synthesis complexity by $1.6 - 2.89\times$ (hundreds of synthesis hours) against the other state of the art frameworks for FPGA DSE. In addition, ATNE is capable of identifying the Pareto designs for certain difficult design spaces which the other existing frameworks are incapable of exploring effectively.

I. Introduction

FPGAs have demonstrated multiple advantages as a heterogeneous accelerator in many different high performance computing tasks [1]–[3]. One of the greatest challenges to implement FPGA accelerated systems is the design complexity due to the nature of hardware design. Recently, system-level (or high-level) synthesis tools have effectively addressed this challenge by replacing the register-transfer level (RTL) design technique with software languages (C++, OpenCL) [4], [5]. The adoption of OpenCL is especially appealing since the language has been widely utilized to program other heterogeneous accelerators such as multi-core CPUs and GPUs. With the state of the art OpenCL synthesis tools, heterogeneous system designers are now able to achieve a longtime desired goal – to program different devices with a single unified language.

Although the OpenCL-to-FPGA tools have significantly reduced the design complexity, one major bottleneck still exists in design space exploration (DSE) due to the extremely long synthesis runtime of these tools. The compilation process of the software OpenCL code running on the processors (CPUs or GPUs) usually only consumes several milliseconds. In contrast to the software compilation, implementing an OpenCL design on the FPGA usually requires multiple hours for a high performance workstation to complete the hardware synthesis (including place and route) process. Moreover, the size of the design space grows exponentially with the number of the tunable parameters in the design. For example, the designer can generate thousands of different designs for a simple matrix multiplication by tuning the parameters in the OpenCL code. Synthesizing all of these designs (demonstrated in Fig. 1) for the brute force DSE is impractical since the process consumes thousands of computing hours.

One solution to this challenge is the machine learning technology which predicts the Pareto-frontier ("good" designs) based on a small training sample. The small training sample reduces the required synthesis time significantly. However, using machine learning is risky. Machine learning may incorrectly exclude some actual Pareto designs in its predicted output. Most of the existing machine learning frameworks attempt to accurately model how the design objective functions respond to the tunable parameters in the designs. This track of effort does not enhance the eventual goal of seeking the "good" designs.

To address this issue, we re-thought how to use machine learning for system level DSE on FPGAs. We propose a framework - *Adaptive Threshold Non-Pareto Elimination* (ATNE) which takes a fundamentally different track from the other existing attempts. **Instead of focusing on improving the conventional accuracy of the learner, our work focuses on understanding and estimating the risk of losing "good" designs due to learning inaccuracy. The goal of ATNE is to minimize this risk.**

Our major contributions are:

- We built a mathematical model to investigate how the machine learning technology should be applied in the system level FPGA DSE task.
- We implemented a novel framework which reduces the synthesis complexity by $1.6-2.89\times$ compared with other state of the art approaches.
- We used 5 end-to-end OpenCL applications (real performance data from applications running on the FPGA,

not just reports from the synthesis tool) to verify the effectiveness of the framework.

To the best of our knowledge, our work is the first attempt to investigate machine learning aided system level (or high-level) DSE with **real performance data from end-to-end (the synthesis process includes the place and route stage) applications running on the FPGA**.

The rest of the paper is organized as follows. We review the state of the art machine learning frameworks for hardware DSE problems in section II. We formulate the problem in section III. In section IV, we provide the theoretical foundation for ATNE. In section V, we describe the ATNE algorithm. We report the evaluation results in section VI. We conclude the paper in section VII.

II. Related Work

In recent years, researchers have been applying machine learning algorithms on hardware DSE problems such as IP core generation [6] and timing results [7]. The high-level design is processed by more layers of the design tool-chain. Moreover, high-level designs are usually written in "software-like" styles. It is more difficult to predict how a high-level parameter would affect the low-level architecture. Therefore, applying machine learning on high-level DSE is a different task than those addressed in the existing low-level tuning frameworks.

Researchers have also attempted to use machine learning to aid high-level DSE. Liu and Carloni proposed a framework using experimental design in [8]. This type of framework focuses on seeking the sampling that describes the design space accurately. The second type of the existing frameworks are uncertainty sampling based. Zuluaga *et al.* proposed an active learning algorithm that iteratively samples the design which the learner cannot clearly classify in [9].

Most of these existing frameworks focus on how to learn an accurate model to describe the design space. The learning accuracy improves the quality of the models that describe how the design space responds to specific tuning parameters. However, this effort may still not be enough to improve the probability of correctly identifying the Pareto-set, which is the ultimate goal of DSE.

Our approach explores a fundamentally different track from these existing works. Instead of further improving the learning accuracy, we chose to understand and estimate the inaccuracy of the learner. Based on the estimated inaccuracy, we adaptively choose a threshold to improve the quality of Pareto design identification.

III. Problem Formulation

In this section, we firstly introduce the concepts of the design space and the Pareto-frontier. Then we formulate the problem that our framework tackles.

Design space and objective space: We refer to the programming choices in the OpenCL code as tunable *knobs*. The examples of knobs and their impacts on the hardware architecture are listed in Table I.

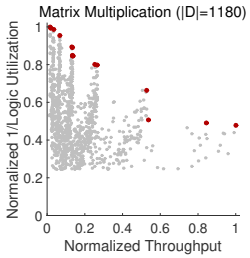

Figure 1. The visualization of the design space of the matrix multiplication application in the objective space using the OpenCL-to-FPGA tool. The red dots are the Pareto-frontier.

Table I
Tunable Knobs in the OpenCL Code and Their Impacts on the Hardware Architecture

Number	OpenCL Knob	Impact on HW Arch.
1	Num. SIMD	SIMD parallel width & mem. BW
2	Num. comp. unit	Num. parallel ctrl. & mem. BW
3	Unroll factor	module replic. mem. access pattern
4	Local buffer height and width	BRAM partition & access pattern
5	Num. private variables	Num. registers
6	Vectorization Width	SIMD parallel width
7	Num. work-items per group	ctrl. logic
8	Data size per work-item	Data locality

For a given application which owns K knobs in the OpenCL code, we use denotation $\mathbf{x} = (x_1, x_2, x_3, ..., x_K)$ to refer to a knob setting. All possible knob settings for a given OpenCL code form $D = \{\mathbf{k}\}$ (called design space). The designers usually evaluate the FPGA designs with multiple objectives (e.g. throughput, logic utilization) which can be represented by functions $\boldsymbol{\phi}(\mathbf{x}) = (\phi_1(\mathbf{x}), \phi_2(\mathbf{x}), \cdots, \phi_M(\mathbf{x}))$: $D \mapsto \mathbb{R}$. Then the objective space is the image of the design space: $\boldsymbol{\phi}(D) \subset \mathbb{R}^M$. Fig. 1 demonstrates the design space image in the objective space of the matrix multiplication application.

Pareto-frontier: In DSE, the goal is to make trade-offs on the Pareto-frontier ("good" designs). Here, we provide the formal definition of the Pareto-frontier. We use canonical partial order "$\prec$" in $\mathbb{R}^M$ to represent one design is inferior in all objectives to another: $\phi(\mathbf{x}) \prec \phi(\mathbf{x}')$ iff $\phi_i(\mathbf{x}) < \phi_i(\mathbf{x}')$, $1 \leq i \leq M$. The Pareto-frontier $P \subseteq D$ is defined as: $P = \{\mathbf{x} \in D \mid \forall \mathbf{x}' \in D, \boldsymbol{\phi}(\mathbf{x}) \succeq \boldsymbol{\phi}(\mathbf{x}')\}$. The red dots in Fig. 1 demonstrate the Pareto-frontier of the matrix multiplication design space.

Problem Formulation: *Given an OpenCL application, predict P synthesizing the minimal number of designs in D.*

Prediction quality: We use *average distance from reference set*(ADRS) [10] as the metric to evaluate the prediction quality of the framework. ADRS is defined as :

$$ADRS(P_{gt}, P_{pred}) = \frac{1}{|P_{gt}|} \sum_{\mathbf{p}_{gt} \in P_{gt}} \min_{\mathbf{p}_{pred} \in P_{pred}} d(\mathbf{p}_{gt}, \mathbf{p}_{pred}),$$

where

$$d(\mathbf{p}_{gt}, \mathbf{p}_{pred}) = \max_{m=0,1,...M} \left(0, \frac{\phi_m(p_{gt}) - \phi_m(p_{pred})}{\phi_m(p_{gt})}\right).$$

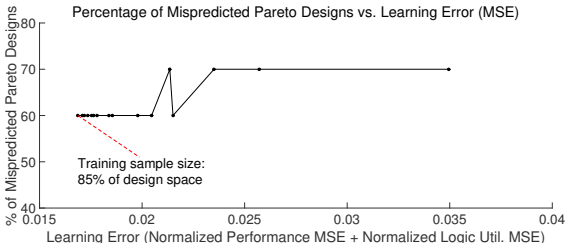

Figure 2. Demonstration of the pitfall of improving learning accuracy. Improving the MSE does not improve the misprediction of Pareto designs.

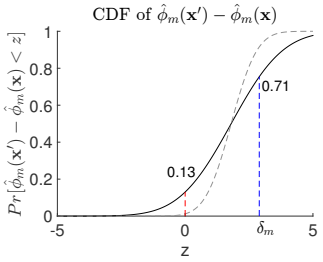

Figure 3. CDF of $\hat{\phi}_m(\mathbf{x}') - \hat{\phi}_m(\mathbf{x})$

We also measure another prediction quality metric, hypervolume error (HVE), as a supplementary evaluation of the framework. The definition of HVE is described in [11].

IV. Re-thinking Machine Learning for System Level FPGA DSE

In this section, we firstly reveal a pitfall in the existing machine learning approaches for FPGA DSE. Then we provide some theoretical results as the foundation of ATNE.

A. Pitfall: attempting to learn more accurately

The main effort of most existing machine learning frameworks in hardware DSE focuses on how to regress the design objective functions more accurately. Improving the regression accuracy certainly generates more precise models to describe the objective function. However, this strategy has a pitfall in the further Pareto design prediction.

Here, we provide empirical results using the matrix multiplication example to demonstrate this pitfall. We used *transductive experimental design* (TED) technology [12] to sample the design space of an OpenCL matrix multiplication code on the FPGA. We used the random forest (RF) algorithm to learn two objective functions – performance and logic resource utilization. We directly use the regressed functions to identify the Pareto designs (i.e. those predicted as inferior to at least one design are considered non-Pareto). We measure two metrics: (1) mean square error (MSE) to illustrate the regression accuracy of the objective functions; (2) percentage of $\mathbf{x}$ such that $\mathbf{x} \in P_{gt}$ and $\mathbf{x} \notin P_{pred}$ to illustrate how many actual Pareto designs are not selected by the learner. We tested multiple training sample sizes (5%, 10%, 15% ... 85% of the entire design space) to generate regressed functions with different accuracies.

As shown in Fig. 2, improving the regression error does not necessarily improve the correctness of the Pareto prediction. Moreover, even when we sampled 85% of the design space to train the learner, the learner still mispredicts approximately 60% of the Pareto points. **This means if we follow the conventional track to focus solely on the learning accuracy, the designer may still miss more than half of the "good" designs!**

B. Re-thinking: understanding the error from the learner

We investigated how the learner inaccuracy affects the final Pareto design prediction quality. We still use RF as an example for the regresser. The RF takes several designs as a training sample to learn the design objective functions. Let $\hat{\boldsymbol{\phi}}$ denote the design objective function vector regressed by the RF. Let T denote the number of decision trees in the RF. The RF algorithm produces the final output by computing an arithmetic mean of the outputs of all the trees:

$$\hat{\boldsymbol{\phi}} = \frac{1}{T}\sum_{t=1}^{T}\hat{\boldsymbol{\phi}}_t.$$

Each tree of the forest is trained on an independently and randomly selected bootstrap of the training data. The outputs of all the trees should independently have the same distribution. Therefore, according to the central limit theorem, the output of the RF is normally distributed (even though we specifically selected RF as the learner here, our theoretical result still stands as long as the output of the learner is an arithmetic mean of i.i.d. random variable). Since the difference between two normally distributed random variables is still normally distributed, for any two designs $\mathbf{x}$ and $\mathbf{x}'$, the difference $\hat{\boldsymbol{\phi}}(\mathbf{x}) - \hat{\boldsymbol{\phi}}(\mathbf{x}')$ is also normally distributed.

With this reasoning, we can describe $\hat{\boldsymbol{\phi}}(\mathbf{x}) - \hat{\boldsymbol{\phi}}(\mathbf{x}')$ as a normally distributed random variable when we apply RF to regress the objective functions of a design space. The expectation and the variance of this random variable are determined by the training data and how we train the forest (the parameter setups of the RF). With the aid of this mathematical model, we can analyze how the regressor (RF) affects the Pareto prediction.

Assuming we use the regressed objective functions to identify the Pareto designs, the conventional threshold is:

$$\mathbf{x} \notin P_{gt}, \text{ if } \exists \mathbf{x}' \in D \text{ s.t. } \hat{\boldsymbol{\phi}}(\mathbf{x}) \prec \hat{\boldsymbol{\phi}}(\mathbf{x}').$$

The cumulative distribution function (CDF) of $\hat{\phi}_m(\mathbf{x}') - \hat{\phi}_m(\mathbf{x})$ is demonstrated in Fig. 3. Assuming $\mathbf{x}$ is incorrectly predicted as inferior to $\mathbf{x}'$ by the RF, the expectation of $\hat{\phi}_m(\mathbf{x}') - \hat{\phi}_m(\mathbf{x})$ should be greater than 0. According to the monotonicity of the normal distribution CDF, we can show that $Pr[\hat{\phi}_m(\mathbf{x}') - \hat{\phi}_m(\mathbf{x}) \leq 0]$ is less than 50%. If variance of $\hat{\phi}_m(\mathbf{x}') - \hat{\phi}_m(\mathbf{x})$ decreases, this probability of correcting the misprediction becomes even lower (shown by the "dot" curve). Therefore, training a more stable learner may decrease the probability for the actual Pareto design to be identified.

An apparent solution to this issue is to lift the threshold from 0 to $\boldsymbol{\delta}$ (a vector of thresholds for multiple design objectives):

$$\mathbf{x} \notin P_{relaxed}, \text{ if } \exists \mathbf{x}' \in D \text{ s.t. } \hat{\boldsymbol{\phi}}(\mathbf{x}) + \boldsymbol{\delta} \prec \hat{\boldsymbol{\phi}}(\mathbf{x}')$$

With this $\boldsymbol{\delta}$, the probability for the learner to think an actual Pareto design $\mathbf{x}$ is not inferior to $\mathbf{x}'$ becomes $Pr[\hat{\phi}_m(\mathbf{x}') - \hat{\phi}_m(\mathbf{x}) \leq \delta_m]$. Due to the monotonicity of the CDF, this new probability is definitely greater than $Pr[\hat{\phi}_m(\mathbf{x}') - \hat{\phi}_m(\mathbf{x}) \leq 0]$. However, lifting this threshold $\boldsymbol{\delta}$ increases the probability for the framework to output those non-Pareto ("bad") designs. For this reason, we name this output set $P_{relaxed}$ meaning "relaxed" Pareto-set. An extreme case of this side-effect is all non-Pareto designs are selected into this $P_{relaxed}$ by lifting the threshold beyond the necessity. Then the framework outputs the entire design space which is equivalent to the brute force approach. Therefore, in order to predict the Pareto designs efficiently, the framework requires an appropriate δ.

We describe this appropriate δ in Theorem 1.

Let μ_m and σ_m denote the expectation and variance of $\hat{\phi}_m(\mathbf{x}') - \hat{\phi}_m(\mathbf{x})$ respectively. Let erf^{-1} denote the inverse Gauss error function (Gauss error function is used in the expression of normal distribution CDF function). We describe our theoretical result as:

Theorem 1: The m component of the minimal threshold $\boldsymbol{\delta}$ to guarantee $Pr[\mathbf{x} \in P_{relaxed} | \mathbf{x} \in P_{gt}] \geq \alpha$ is

$$\delta_m = \max_{\forall \mathbf{x} \in P_{gt},\, \forall \mathbf{x}' \in D} (\mu_m + \sigma_m \sqrt{2} erf^{-1}(2\alpha - 1)).$$

The proof of Theorem 1 is straightforward – use the quantile formula of the normal distribution and its monotonicity. We describe how to practically estimate this minimal threshold in lines 8 to 17 of Algorithm 1.

V. ATNE Algorithm

We describe ATNE in algorithm 1. ATNE actively samples data and iteratively eliminates the non-Pareto designs. Although ATNE uses two existing technologies, active learning and experimental sampling, we applied these technologies with a completely novel strategy. The major novelty of the ATNE algorithm is the three stages (lines 8 to 33 in Algorithm 1): (1) estimate $\boldsymbol{\delta}$, (2) elimination and (3) active sampling. The stages initial sampling and model learning only serve as the starting point for the other three stages. We describe each stage in detail in the rest of this section.

Initial Sampling: At the initial state, we use TED to sample the designs purely based on the knob setting information. TED has been proven effective for this task in [8]. This initial stage only serves as a starting point for the $\boldsymbol{\delta}$ estimation and elimination stages. We only sample 6 designs (minimum for the first round of random forests regression).

Model Learning (Regression): The random forest is suitable for the discrete nature of the design knob settings. Therefore, we chose RF for regression. As described in lines 5 to 7 of Algorithm 1, we train multiple RFs with the sampled designs instead of a single forest. Using these trained forests, we can estimate the μ and σ of the distribution of $\hat{\phi}_m(\mathbf{x}') - \hat{\phi}_m(\mathbf{x})$ for the next stage. Also, the regressed objective functions are used for eliminating the non-Pareto designs.

Estimate $\boldsymbol{\delta}$: Estimating an appropriate $\boldsymbol{\delta}$ aids the framework to find the Pareto designs accurately and efficiently. The theoretical minimal $\boldsymbol{\delta}$ is provided by Theorem 1. However, in reality, the framework cannot obtain the real distribution as in the theorem. Therefore, we use the sampled data to estimate this $\boldsymbol{\delta}$. We compute the approximate μ and σ from the regressed results of the F forests as shown in lines 10 to 12 of the algorithm. Among all the sampled data, we only collect those "under-estimated" (i.e. the forests predict this design is inferior to another design when in fact the ground truth design is non-inferior, as shown in line 13) ones to compute the $\boldsymbol{\delta}$ candidates. In order to maximize the

Algorithm 1: The ATNE Algorithm

Input : Design space D; Number of initial samples S_{init}; Target final size of $P_{relaxed}$: S_{final}; Target correctness probability α; Number of forests F; Minimal number of candidates for $\boldsymbol{\delta}$ estimation: C

1 $P_{relaxed} = D$, $L = \emptyset$

Initial Sampling

2 initially sample the designs L=TED(D, S_{init})

3 **while** $|P_{relaxed}| < S_{final}$ **do**

4 set of designs to be eliminated $E = \emptyset$

Model Learning

5 **for** $f = 1$ *to* F **do**

6 regress $\hat{\boldsymbol{\phi}}_f$ on L using random forest f

7 **end**

Estimate $\boldsymbol{\delta}$

8 $\Delta_{candi.} = \emptyset$

9 **for** $\forall \boldsymbol{x}$ *and* $\forall \boldsymbol{x}' \in P_{relaxed} \cap L$ **do**

10 $\hat{\mu}_{\mathbf{x},\mathbf{x}'} = \frac{1}{F} \sum_{f=1}^{F} \hat{\boldsymbol{\phi}}_f(\mathbf{x}') - \hat{\boldsymbol{\phi}}_f(\mathbf{x})$

11 $\hat{\sigma}_{\mathbf{x},\mathbf{x}'} = \sqrt{\frac{1}{F} \sum_{f=0}^{F} (\hat{\boldsymbol{\phi}}_f(\mathbf{x}') - \hat{\boldsymbol{\phi}}_f(\mathbf{x}) - \hat{\mu}_{\mathbf{x},\mathbf{x}'})^2}$

12 $\boldsymbol{\delta}_{\mathbf{x},\mathbf{x}'} = \hat{\mu}_{\mathbf{x},\mathbf{x}'} + \hat{\sigma}_{\mathbf{x},\mathbf{x}'} \sqrt{2} erf^{-1}(2\alpha - 1)$

13 **if** $\hat{\mu}_{\boldsymbol{x},\boldsymbol{x}'} > 0$ *and ground truth* $\boldsymbol{x}$ *is superior to* $\boldsymbol{x}'$ **then**

14 $\Delta_{candi.} = \Delta_{candi.} \cup \{\boldsymbol{\delta}_{\mathbf{x},\mathbf{x}'}\}$

15 **end**

16 **end**

17 $\boldsymbol{\delta} = \max(\Delta_{candi.})$

Elimination

18 $E = \emptyset$

19 **for** $\forall \boldsymbol{x}$ *and* $\forall \boldsymbol{x}' \in P_{relaxed}$ **do**

20 **if** $\hat{\boldsymbol{\phi}}_f(\boldsymbol{x}') - \hat{\boldsymbol{\phi}}_f(\boldsymbol{x}) \prec \boldsymbol{\delta}$ *and* $|\Delta_{candi.}| \geq C$ **then**

21 $E = E \cup \{\mathbf{x}\}$

22 **end**

23 **end**

24 $P_{relaxed} = P_{relaxed} - E$

Active Sampling

25 $\Delta_{sample} = \emptyset$

26 **for** $\forall \boldsymbol{x}$ *and* $\forall \boldsymbol{x}' \in P_{relaxed} - (P_{relaxed} \cap L)$ **do**

27 compute $\boldsymbol{\delta}_{\mathbf{x},\mathbf{x}'}$ similarly to lines 10 to 12

28 **if** $\hat{\mu}_{\boldsymbol{x},\boldsymbol{x}'} < 0$ **then**

29 store $\boldsymbol{\delta}_{\mathbf{x},\mathbf{x}'}$ in array $A_{\boldsymbol{\delta}}$

30 **end**

31 **end**

32 read all $\boldsymbol{\delta}_{\mathbf{x},\mathbf{x}'}$ from $A_{\boldsymbol{\delta}}$, compute $\hat{\mathbf{x}} = \arg\min_{\mathbf{x}} \sum_{\forall \mathbf{x}' \in P_{relaxed} - (P_{relaxed} \cap L)} \boldsymbol{\delta}_{\mathbf{x},\mathbf{x}'}$

33 synthesize $\hat{\mathbf{x}}$, $L = L \cup \{\hat{\mathbf{x}}\}$

34 **end**

35 synthesize designs in $P_{relaxed} - (P_{relaxed} \cap L)$

36 P_{pred} = brute force Pareto-frontier search on $P_{relaxed} \cup L$

Output : predicted Pareto set P_{pred}

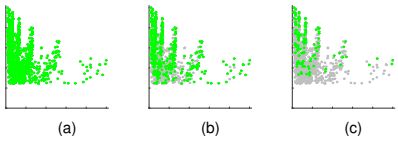

Figure 4. Transition of eliminating the predicted non-Pareto designs. The grey dots represent the eliminated designs (set E). The green dots represent set $P_{relaxed}$). (a) initial state; (b) iteration = 15; (c) iteration = 80.

probability $Pr[\mathbf{x} \in P_{relaxed} | \mathbf{x} \in P_{gt}]$), we then select the greatest candidate to be the estimated $\boldsymbol{\delta}$.

Elimination: The elimination happens only when the number of $\boldsymbol{\delta}$ candidates is greater than the parameter C (shown in line 20). This prevents the case when the framework does not have enough ground truth data to estimate $\boldsymbol{\delta}$. When the framework has enough ground truth to estimate $\boldsymbol{\delta}$, it eliminates the designs that meet the criteria $\hat{\phi}_f(\mathbf{x}') - \hat{\phi}_f(\mathbf{x}) \prec \boldsymbol{\delta}$. As discussed earlier, using this criteria, the algorithm is unlikely to eliminate the ground truth Pareto designs. The size of $P_{relaxed}$ iteratively decreases. The transition of this elimination process is visualized in Fig.4. As shown in this example, the eliminated designs are mostly non-Pareto in each iteration.

Active Sampling: The traditional active sampling aims to provide the data to refine the learner [9], [13]. However, as discussed in section IV, this traditional strategy may not serve the DSE goal effectively. In contrast to traditional active sampling, our strategy in ATNE focuses on mitigating the side-effect that $\boldsymbol{\delta}$ increases $Pr(\mathbf{x} \in P_{pred} | \mathbf{x} \notin P_{gt})$. As shown in lines 24 to 33 of Algorithm 1, we select the design that has the highest $Pr(\mathbf{x} \in P_{pred} | \mathbf{x} \notin P_{gt})$. Therefore, in order to eliminate this design quickly, we synthesize it to enable its elimination by using the ground truth data.

VI. Results

We collected data from 5 end-to-end OpenCL applications running on the actual FPGA. The synthesis processes of these applications include the place and route stage. Collecting the ground truth data of all 5 design spaces costs us more than 10000 computing hours (5 months using 3 high-end 8-core workstations). We use these ground truth data to evaluate the prediction quality of ATNE. In the rest of this section, we firstly describe the experimental setup. Secondly, we compare the prediction quality of ATNE against the other state of the art frameworks.

A. Experimental Setup

Our benchmarking OpenCL applications are matrix multiplication, sobel filter, FIR filter, histogram and DCT. As listed in table II, these benchmarks cover several computation patterns that are often used in other high performance computing applications. The benchmarks also cover several widely used OpenCL programming techniques. The specifications (such as input image resolution, filter tap size, etc.) of these applications are also listed in the table. The listed tunable knobs are represented by numbers correspondent to the first column in table I.

Table II
Benchmarking Applications

Benchmark	Basic Comp.	OpenCL Tech.	Spec.	Knobs
MM	matrix op.	blocking	1024×1024	1 - 8
Sobel	sliding window	blocking	1080p	1,2,4-8
FIR	sliding tap	streaming	128 taps	1,2,4,5,7,8
HIST	global sum	blocking	256 bins	2-5,7,8
DCT	transformation	blocking	8×8 block	1 - 8

The OpenCL-to-FPGA tool we used is Altera OpenCL SDK 14.1. The experimental board is Terasic DE5-net with an Altera Stratix V FPGA. We choose the two most important design metrics – throughput and logic utilization (ALMs) as the learning objectives.

We configure the parameters of ATNE as follows: we set $S_{init} = 6$ (only as a starting point, the main sampling process relies on the active learning stage of ATNE); we set the target size $S_{final} = 40$ as a reasonable synthesizing time. We empirically set the parameter C to 10. We set the number of forests $F = 200$. We ran ATNE with multiple α's to evaluate the prediction qualities of ATNE for different synthesis costs. The RF algorithm we used is the Matlab MEX version converted from the Fortran version designed by Breiman [14]. We configured the number of trees to 50 and the bootstrap coefficient to 0.37 for the RF.

B. Prediction Quality

Fig. 5 visualizes the Pareto prediction results produced by ATNE. The synthesis complexities required to produce these results are also listed in Fig. 5. The framework identified most of the Pareto designs.

We quantify the prediction quality of ATNE using ADRS calculated by the normalized ground truth data. We compare the ADRS qualities of ATNE against the other two state of the art frameworks PAL [9] and TED [8]. As shown in Fig. 6, we report the ADRS vs. sampling complexity of all three frameworks. From the figure, it is obvious that ATNE outperforms the other two frameworks. More specifically, we setup an accuracy threshold ADRS$\leq$ 0.01 meaning the predicted Pareto points are inferior to the ground truth Pareto points no more than 1% for any design objective. For a real-world design task, this means we find a design that is almost the optimal. The minimal sampling complexities required to reach the threshold are listed in table III. Compared with PAL and TED, ATNE achieves a $1.6\times$ synthesis speedup for MM, $2.23\times$ for Sobel, $2.89\times$ for Hist. Since each synthesis run takes hours, these speedups could save hundreds of computing hours for the designers. None of the frameworks reach ADRS$\leq$ 0.01 for FIR. The best ADRS results and their correspondent synthesis complexities for ATNE, TED and PAL are 0.0155 for 21.58%, 0.0356 for 22.45% and 0.0796 for 30.71% respectively. Therefore, ATNE still achieves the best prediction quality with the least synthesis complexity for FIR.

We also report the prediction quality in HVE as a supplementary evaluation. As shown in table IV, for all 5 benchmarks, ATNE still consumes the least synthesized samples

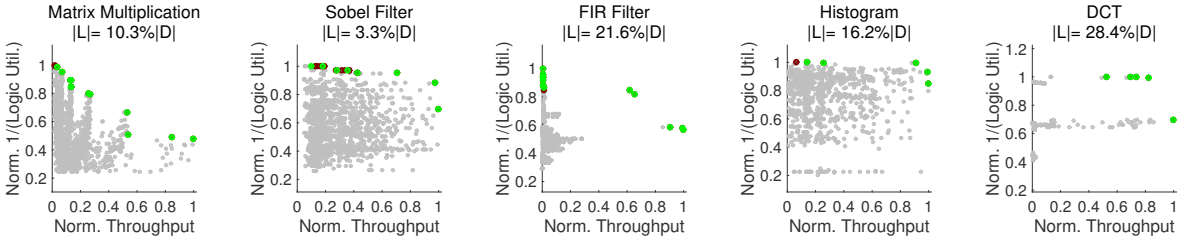

Figure 5. Demonstration of predicted Pareto Points. The Pareto prediction results produced by ATNE with ADRS$\leq$ 0.01 (ADRS=0.015 for FIR) are demonstrated. The green dots represent the predicted Pareto designs. The red dots represent the ground truth designs that are not identified by the framework.

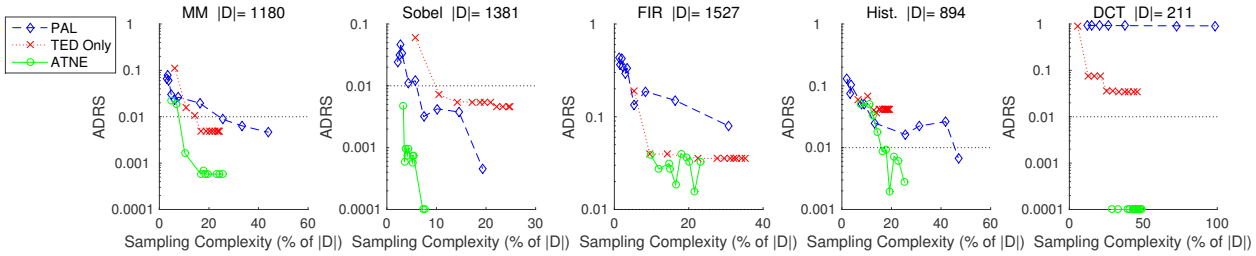

Figure 6. The comparison between prediction qualities (ADRS) of ATNE and the other state of the art approaches PAL [9], TED only [8]. We ran ATNE on $\alpha = 0.6, 0.8, 0.9, 0.95, 0.9750, 0.9875, 0.9938, 0.9969, 0.9984, 0.9992$. The vertical axes are in logarithmic scale.

Table III
MINIMAL SAMPLING COMPLEXITIES TO REACH THE PREDICTION QUALITY (ADRS) THRESHOLD

	MM	Sobel	FIR	Hist	DCT
ATNE	10.25%	3.33%	NA	16.24%	28.44%
TED	16.72%	10.55%	NA	NA	NA
PAL	25.56%	7.43%	NA	46.96%	NA

Table IV
MINIMAL SAMPLING COMPLEXITIES TO REACH THE PREDICTION QUALITY (HVE) THRESHOLD

	MM	Sobel	FIR	Hist	DCT
ATNE	10.25%	3.33%	9.72%	19.24%	28.44%
TED	14.14%	14.12%	22.45%	NA	NA
PAL	44.07%	7.43%	NA	46.96%	NA

to reach the prediction quality HVE$\leq$ 0.01 among the three frameworks.

VII. CONCLUSION

In this paper, we presented a machine learning framework – ATNE for the design space exploration on the OpenCL-to-FPGA tool. ATNE applies a novel strategy of machine learning on the system level FPGA DSE task. We evaluated the effectiveness of ATNE using 5 end-to-end OpenCL applications running on the actual FPGA board. The experimental results indicate that ATNE outperforms the other state of the art frameworks by $1.6 - 2.89\times$ in sampling complexity for the same prediction accuracy. ATNE is also capable of identifying the Pareto designs for the difficult design spaces which the other existing frameworks are incapable of exploring effectively.

REFERENCES

[1] D. Hefenbrock *et al.*, "Accelerating viola-jones face detection to fpga-level using gpus," in *Proc. 18th IEEE Annu. Int. Symp. Field-Programmable Custom Computing Machines (FCCM)*, 2010, pp. 11–18.

[2] C. Olson *et al.*, "Hardware acceleration of short read mapping," in *Proc. 20th IEEE Annu. Int. Symp. Field-Programmable Custom Computing Machines (FCCM)*, 2012, pp. 161–168.

[3] C. Zhang *et al.*, "Optimizing fpga-based accelerator design for deep convolutional neural networks," in *Proc. ACM/SIGDA Int. Symp. Field-Programmable Gate Arrays (FPGA)*, 2015, pp. 161–170.

[4] D. Chen and D. Singh, "Fractal video compression in opencl: An evaluation of cpus, gpus, and fpgas as acceleration platforms," in *Proc. 18th Asia and South Pacific Design Automation Conf. (ASP-DAC)*, 2013, pp. 297–304.

[5] V. Morales *et al.*, "Energy-efficient fpga implementation for binomial option pricing using opencl," in *Proc. Conf. on Design, Automation & Test in Europe (DATE)*, 2014, pp. 208:1–208:6.

[6] M. K. Papamichael *et al.*, "Nautilus: Fast automated ip design space search using guided genetic algorithms," in *Proc. 52nd Annu. Design Automation Conf. (DAC)*, 2015, pp. 43:1–43:6.

[7] N. Kapre *et al.*, "Intime: A machine learning approach for efficient selection of fpga cad tool parameters," in *Proc. ACM/SIGDA Int. Symp. Field-Programmable Gate Arrays (FPGA)*, 2015, pp. 23–26.

[8] H.-Y. Liu and L. P. Carloni, "On learning-based methods for design-space exploration with high-level synthesis," in *Proc. 50th Annu. Design Automation Conf. (DAC)*, 2013, pp. 50:1–50:7.

[9] M. Zuluaga *et al.*, "Active learning for multi-objective optimization," in *Proc. 30th Int. Conf. on Machine Learning (ICML)*, 2013, pp. 462–470.

[10] G. Palermo *et al.*, "Respir: A response surface-based pareto iterative refinement for application-specific design space exploration," *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, vol. 28, no. 12, pp. 1816–1829, Dec 2009.

[11] E. Zitzler *et al.*, "The hypervolume indicator revisited: On the design of pareto-compliant indicators via weighted integration," in *Proc. 4th Int. Conf. on Evolutionary Multi-criterion Optimization*, 2007, pp. 862–876.

[12] K. Yu *et al.*, "Active learning via transductive experimental design," in *Proc. 23rd Int. Conf. Machine Learning (ICML)*, 2006, pp. 1081–1088.

[13] S. Dasgupta, "Two faces of active learning," *Theor. Comput. Sci.*, vol. 412, no. 19, pp. 1767–1781, Apr 2011.

[14] L. Breiman, "Random forests," *Machine Learning*, vol. 45, no. 1, pp. 5–32, Oct 2001.