AutoShrink: Adaptive Search Space Shrinkage for Large-Scale Pareto Optimization of HLS Designs

Yingxin Zeng[1], Binghao Cheng[1], Jianwang Zhai[2], Kang Zhao[2], Zhe Lin[1†]
[1]Sun Yat-sen University, Shenzhen Campus, China [2]Beijing University of Posts and Telecommunications, China
zengyx29@mail2.sysu.edu.cn, chengbh5@alumni.sysu.edu.cn, {zhaijw, zhaokang}@bupt.edu.cn, linzh235@mail.sysu.edu.cn

***Abstract*—High-level synthesis (HLS) streamlines accelerator customization by delivering a high-level hardware programming paradigm enriched with a variety of optimization directives. However, the quality of HLS designs is largely determined by the selection of directives in navigating trade-offs among multiple design metrics, a non-trivial process that can significantly prolong design turnaround time. Design space exploration (DSE) serves as a promising solution to this problem, but existing studies on DSE suffer from a lack of efficiency or generalization capability in large-scale application scenarios. To address this problem, this paper proposes *AutoShrink*, a DSE engine that automatically and adaptively shrinks the large search space of an HLS design to gradually retain only high-quality solutions. AutoShrink incorporates: (1) a comprehensive design space pruning strategy that integrates domain knowledge and consolidates the joint effect of directives; and (2) an importance-guided Pareto optimization algorithm that dynamically tracks the importance ranking of the applied directives and leverages this ranking to effectively steer the search toward Pareto-optimal solutions. Experimental results demonstrate that AutoShrink efficiently achieves a close approximation of the Pareto frontier across diverse benchmarks with design spaces scaling up to 10^{16}, which attains an average deviation of only 8.1%, outperforming three generic optimization methods and three state-of-the-art customized approaches by 5.73× and 4.47×, respectively.**

***Index Terms*—High-level synthesis, design space optimization, quality of results.**

I. Introduction

High-level synthesis (HLS) [1] enables designers to describe hardware behavior using high-level languages such as C or C++. Compared to the traditional register-transfer level (RTL) workflow, HLS design methodology delivers a salient speedup of more than 6× [2]. Modern HLS tools [3]–[5] provide designers with a series of optimization directives to rapidly tune the hardware micro-architectures without modifying the C/C++ source code. The quantity and diversity of HLS directives constitute a vast configuration space, wherein the selection and configuration of each directive can yield a distinct hardware implementation, thereby influencing the quality of results (QoR). In most cases, HLS design optimization relies heavily on domain expertise to guide directive selection, typically involving a time-consuming iterative process of directive tuning and quality verification until a satisfactory QoR trade-off is reached.

Design space exploration (DSE) [6] has emerged as an effective and concise solution to the problem of automatically finding high-quality directive configurations that meet designers' QoR requirements from a vast number of directives. At the heart of DSE is the construction of a surrogate model to estimate the QoR for unexplored design instances, based on which the most promising candidates can be identified for future exploration. Some research works [7]–[13] build up surrogate models in an online fashion. That is, a dedicated surrogate model is constructed from scratch as per the target application, and it is kept calibrated using synthesis results of newly explored design instances. However, developing a high-quality surrogate model usually necessitates the accumulation of a substantial number of representative design instances, which incurs large runtime overhead from cold-start conditions with insufficient data. This problem is even exacerbated in large design spaces. In contrast, another branch of DSE methods [14]–[17] aims to establish a predictive model that is generally applicable to various designs and uses it as a well-defined surrogate model without the need for further refinement, thereby avoiding time-consuming online calibration and ensuring high efficiency. Nevertheless, these pre-trained methods require the construction of high-quality training datasets in advance, and their prediction models often exhibit low accuracy in new application domains, thus revealing poor generalization ability.

Overall, existing HLS DSE methods are constrained by a critical dilemma: when confronted with large and intricate design spaces, they either suffer from low exploration efficiency or demonstrate limited generalization capability. To address this challenge, this paper strives to put forward an efficient yet generalizable DSE framework, specifically targeting large-scale design spaces. The core idea of our method originates from two central observations. *Observation 1: The interaction between directives can sometimes result in a large number of redundant or under-optimized configurations, particularly for large design spaces.* These situations can be identified and pruned away in advance with the guidance of HLS domain knowledge and designers' experience. *Observation 2: Directives exert varying degrees of influence on the QoR of a given application.* Pinpointing the most influential directives and fully exploiting their potential helps guide the search toward faster convergence to optimal solutions.

Building upon this foundation, we propose *AutoShrink*, a novel HLS DSE engine that automatically and adaptively shrinks the search space into compact yet high-quality subspaces at both *pre-exploration* and *intra-exploration* stages. To the best of our knowledge, AutoShrink is the first approach to achieve both efficient and generalizable multi-objective HLS DSE in design spaces of up to 10^{16} configurations. The contributions of AutoShrink are summarized as follows:

- We propose AutoShrink, a DSE framework that can be

†Corresponding author.

978-3-9826741-1-7/DATE26/© 2026 EDAA

seamlessly integrated into existing DSE flows to guide the Pareto search through design space shrinkage, demonstrating high efficiency and strong generalization ability.
- We introduce a comprehensive design space pruning strategy that eliminates redundant and suboptimal design points through the joint exploitation of directive interactions and domain knowledge.
- We describe an importance-guided exploration algorithm, which adaptively deduces the importance ranking of directives and progressively narrows the scope of exploration to a reduced yet promising subset of solutions.
- Experimental results show that AutoShrink attains an average distance to reference set (ADRS) of 8.1% across a wide range of applications with design spaces spanning 10^3 to 10^{16}, achieving improvements of 5.73× and 4.47× over generic optimization and state-of-the-art (SOTA) DSE methods, respectively.

II. The AutoShrink Framework

A. Problem Formulation

Given a C/C++ application associated with a set of HLS directives, the objective of our HLS DSE is to approach the *Pareto frontier* P, i.e., the set of optimal design points for multiple QoR metrics, including latency and the resource utilization of look-up table (LUT), block random access memory (BRAM), and digital signal processing (DSP). The DSE iteratively samples promising design points, which collectively form the *approximate Pareto frontier* $\hat{P}$.

The *ADRS* metric is adopted to evaluate the quality of DSE results. Specifically, ADRS quantifies the deviation between P and $\hat{P}$, which is formally defined as:

$$ADRS(\hat{P}, P) = \frac{1}{|P|} \sum_{p \in P} \min_{\hat{p} \in \hat{P}} f(\hat{p}, p), \tag{1}$$

$$f(\hat{p}, p) = \max \left\{ 0, \frac{l(\hat{p}) - l(p)}{l(p)}, \frac{t(\hat{p}) - t(p)}{t(p)}, \frac{m(\hat{p}) - m(p)}{m(p)}, \frac{d(\hat{p}) - d(p)}{d(p)} \right\}, \tag{2}$$

where $l(\cdot)$, $t(\cdot)$, $m(\cdot)$, $d(\cdot)$ are functions to retrieve the latency and the resource usage of LUT, BRAM and DSP, respectively.

B. Overall Framework

As shown in Fig. 1 (left), AutoShrink adheres to a typical DSE flow without relying on pre-developed surrogate models. This DSE flow begins by randomly sampling a subset of design points, extracting their QoR metrics via HLS, and initializing the surrogate model. Then, the surrogate model and a design point sampling engine work collectively to identify the most promising candidate to explore in the next round. This candidate passes through HLS to obtain QoR metrics, with which the surrogate model and the approximate Pareto frontier are updated. The above steps constitute an *exploration–evaluation–feedback* loop, which iterates until the predefined time budget is reached, ultimately producing the approximate Pareto frontier $\hat{P}$. Without loss of generality, we utilize the widely used and high-performance Bayesian optimization to provide both the surrogate model and the sampler in the flow.

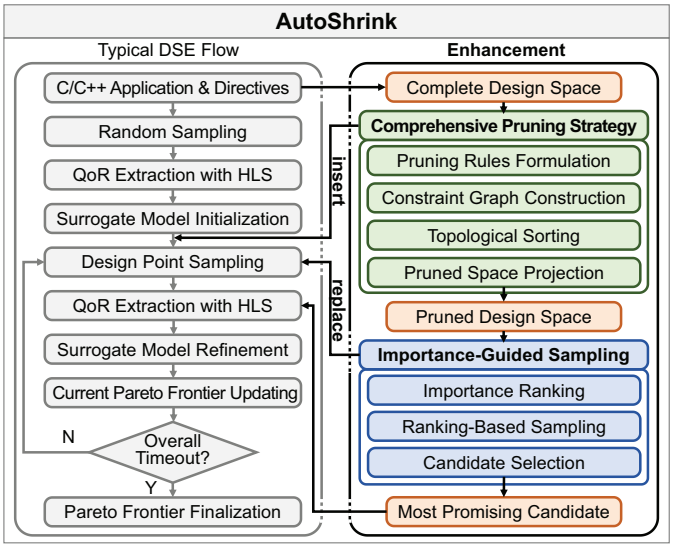

Fig. 1. AutoShrink features two components, comprehensive pruning and importance-guided sampling, which integrate easily into typical DSE flows for large-scale scenarios.

Building on this basic flow, AutoShrink seamlessly integrates two customized optimization strategies into the DSE routine, while ensuring generality and compatibility with existing methods, as depicted in Fig. 1 (right). Together, these two key components steer the design search throughout two phases: a comprehensive pruning strategy is inserted during *pre-exploration* to significantly reduce the design space to examine; an importance-guided sampling replaces the primitive sampling engine and enhances candidate selection during *intra-exploration*.

III. Comprehensive Pruning Strategy

Motivated by *observation 1* in Section I, AutoShrink begins by constructing a compact but effective design space. We notice that prior studies [18], [19] have introduced design space pruning techniques for HLS DSE. However, each of these studies is predominantly guided by a simple rule, which proves effective for small design spaces but fails to scale to larger ones. In light of this, we derive a comprehensive set of pruning rules targeting key directives for functions, loops, and arrays. On this basis, we further develop an enhanced pruning strategy capable of efficiently narrowing large design spaces.

A. Pruning Rules Formulation

We generalize three types of rules from a variety of directives by carefully profiling multiple applications with Vitis HLS [3]. Each rule identifies a directive that constrains the configuration of one or more directives. In HLS, directives take effect in a top-down manner: from the top function to inner functions, and subsequently to loops and arrays. Accordingly, we establish the constraining relationships of directives in our rules following the same top-down order, as shown in Table I. Without loss of generality, our pruning rules can be easily extended to other mainstream HLS tools such as LegUp [4] and Bambu [5], as our focus is on generic program constructs and HLS principles that are consistent across different tools.

Rule-1: The adoption of some directives precludes the specification of other directives. *(1) Outer functions'*

TABLE I
PRUNING RULES AND DIRECTIVES' RELATIONSHIPS.

Rule	Constraining Directive	Configuration	Constrained Directive	Configuration
1.1	Function Dataflow	On	Inner Functions' Inline	Off
1.2	Loop Unroll	Partially Unroll	Inner Loops' Flatten	Off
1.3	Loop Pipeline	On	Inner Loops' Unroll	Fully Unroll
2.1	Function Inline	On	Same Function's Dataflow	Off
2.2	Loop Unroll	Fully Unroll	Same Loop's Flatten	Off
			Same Loop's Pipeline	Off
3.1	Loop Unroll (Access Same Array)	List of Unroll Factors	Accessed Array's Partition	Max of Unroll Factors
3.2	Loop Merge	On	Inner Functions' Inline	On

dataflow disables inner functions' inlining: Function dataflow overlaps the execution of inner functions, which requires preserving their boundaries. This prohibits the inlining of inner functions, as it embeds their bodies into the parent and removes such boundaries. *(2) Partial unrolling of an outer loop invalidates inner loop flattening*: Partial unrolling of an outer loop duplicates inner loop structures, producing a non-perfectly nested loop. This conflicts with the flatten directive applied to the inner loop, which requires perfect nesting to fuse multiple nested loops into a single flat loop. *(3) Outer loop pipelining enforces full unrolling of inner loops*: Outer-loop pipelining aims to schedule all operations holistically across the loop body. Accordingly, inner loop structures are eliminated through complete unrolling, rendering any further unrolling configurations in vain.

Rule-2: C/C++ components can be removed by certain directives, causing other directives on the same component to be ineffective. We observe that certain directives will reorganize specific program constructs, thereby influencing the applicability of other directives targeting the original structures. *(1) Function inline invalidates directives on the same function*: When a function is inlined, its body is merged into the parent function, eliminating the original function instance and causing all directives that are applied to that function to become invalid. *(2) Full loop unrolling prohibits other directives from being applied at the same loop level*: When a loop is fully unrolled, it is transformed into discrete operations, and the loop itself no longer exists. Therefore, directives like loop pipelining and flattening that previously operated on this loop are rendered ineffective.

Rule-3: Specific directives should be coordinated to achieve performance gain. *(1) Loop unroll factor should match the partition factor of accessed arrays*: Loop unrolling increases parallelism along with memory bandwidth demand. Optimal performance is generally achieved when the partition factor applied to an array dimension matches the loop unroll factor used to access that dimension. Otherwise, parallelism from loop unrolling is limited by insufficient memory bandwidth. *(2) Loop merging across functions requires function inlining*: Loop merging is possible only when the target loops reside at the same code hierarchy. When loops are located in different functions, function inlining is typically required to bring them into the same scope. For example, inlining $\{function1\{loop1\}, function2\{loop2\}\}$ allows $loop1$ and $loop2$ to be merged under a $loop\_merge$ directive.

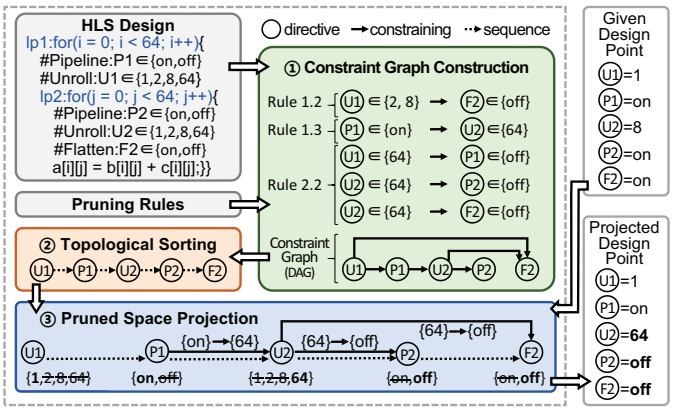

Fig. 2. A simplified example of the comprehensive pruning strategy.

B. Pruning Strategy

The aforementioned pruning rules define intricate constraints among directives. We propose a strategy to orchestrate these rules so as to transform an arbitrary design point into a valid one that satisfies all applicable rules, thereby yielding a reduced design space for each application. This pruning strategy consists of three stages.

Constraint graph construction. Given an HLS design of interest, we first extract all directives applied to this design, and construct a *constraint graph* $G = (V, E)$ that depicts the complete directive set, wherein each node $v \in V$ is a directive instance in the directive set and each edge $e \in E$ represents a directed relation from the constraining directive to the directive being constrained as defined in Table I.

Topological sorting. Since the relationships among directives in the pruning rules are established in a top-down order, the edges of the constraint graph consistently follow the same order, pointing from functions to loops and then to arrays. This prevents any reverse edges from inner to outer hierarchies, ensuring that the constraint graph is a directed acyclic graph (DAG). On this foundation, we perform *topological sorting* [20] on the constraint graph, which is only applicable to DAGs and returns an ordered list of nodes, ensuring that the sink nodes are located after their source nodes. In other words, passing through the ordered list ensures strict adherence to the pruning rules defined by the constraint graph.

Pruned space projection. After obtaining the topologically sorted list of directives, we seek to project an arbitrary design point into the pruned space. Considering the applicable directives, their specified configurations and all possible configurations, we traverse the sorted list of directives sequentially and specify their configurations under the pruning rules gradually. Unconstrained directives can retain their original configurations. However, for constrained directives, we filter out invalid configurations according to pruning rules, keep only the feasible subset, and finally choose the configuration closest to the pre-set one based on cosine similarity. This process dynamically projects a design point into the pruned space rather than enumerating all valid design points, which

significantly reduces computational cost and supports fast pruning for large-scale design spaces at runtime.

For ease of understanding, we provide a simplified example in Fig. 2. Given an arbitrary design point $\langle U1 = 1,\ P1 = on,\ U2 = 8,\ P2 = on,\ F2 = on\rangle$, we identify the applicable pruning rules and build the constraint graph with five directive nodes and five edges. As the graph is acyclic, we perform a topological sort on the directive nodes and then apply the pruned space projection. In this order, $U1$ and $P1$ remain unchanged; $P1 = on$ forces $U2$ to be fully unrolled with the factor of 64, which eliminates $loop2$ and makes both $P2$ and $F2$ inapplicable (both set to off). Finally, the projected design point becomes $\langle U1 = 1,\ P1 = on,\ U2 = 64,\ P2 = off,\ F2 = off\rangle$.

IV. Importance-Guided Sampling

In the exploration stage, existing DSE methods [7]–[13] typically rely on simple sampling strategies such as random sampling to find a subset of design points as promising candidates. This practice renders large-scale DSE extremely slow, as high-quality design points are rarely identified due to the low probability of random selection. To address this limitation, we propose substituting the sampling engine with an enhanced exploration algorithm, which comprises three steps: *importance ranking*, *ranking-based sampling* and *candidate selection*, as depicted in Algorithm 1.

Importance ranking. After profiling a large number of HLS applications, we observe that directives exert varying degrees of influence on the final QoR of the synthesized hardware. This motivates us to develop a sampling engine capable of discerning and leveraging the unique impact of each directive on QoR. To start with, we employ the functional analysis of variance (fANOVA) [21]. Specifically, fANOVA decomposes a multivariate function $f(\mathbf{x})$ with $\mathbf{x} = (x_1, x_2, \ldots, x_n)$ into additive components that capture the effects of individual variables and their interactions:

$$f(\mathbf{x}) = f_0 + \sum_{i=1}^{n} f_i(x_i) + \sum_{i<j} f_{ij}(x_i, x_j) + \cdots + f_{1,2,\ldots,n}(x_1, x_2, \ldots, x_n). \tag{3}$$

In our context, each inserted HLS directive is treated as a distinct variable x_i (categorical values such as on/off are encoded numerically, e.g., as 1 and 0), and the target function $f(\mathbf{x})$ corresponds to an approximation of QoR metrics considered in this work, i.e., latency and the utilization of LUT, BRAM, and DSP. In practice, this function is usually implemented as a random forest that is iteratively updated during exploration [21]. Based on this, the importance of an individual directive x_i can be quantified as the ratio of the variance attributed to its impact on QoR to the total variance of QoR across all cases, which can be represented as:

$$I_i = \frac{\mathrm{Var}[f_i(x_i)]}{\mathrm{Var}[f(\mathbf{x})]}, \tag{4}$$

where the variance of directive i's impact on QoR, $\mathrm{Var}[f_i(x_i)]$, can be defined as

$$\mathrm{Var}[f_i(x_i)] = \frac{1}{\|\mathbf{\Theta}_i\|} \int f_i^2(x_i)\, dx_i, \tag{5}$$

Algorithm 1: Importance-Guided Sampling

Input: P, P_{last}: current/last-round Pareto frontier; S_p: flag for using pruned space (initially true); N: number of samples per round; $cr_{\min}$: minimum change rate; $T, T_{\max}$: counter of consecutive rounds without Pareto frontier improvement
Output: Most promising candidate $\mathbf{x}^*$

1 // Stage 1: Importance Ranking
2 Estimate importance scores $[I_1, I_2, \ldots, I_n]$ for directives $[x_1, x_2, \ldots, x_n]$ using fANOVA (Eq. 4);
3 $R \leftarrow$ list of directives ranked by importance scores (descending);
4 // Stage 2: Ranking-Based Sampling
5 Initialize $[\mathbf{x}] \leftarrow [\,]$ (list of sampled design points);
6 **for** $j = 1$ **to** N **do**
7 Pick $\mathbf{x} \in P$ at random;
8 **for** $k = 1$ **to** $|R|$ **do**
9 $cr_k \leftarrow cr_{\min} + (1 - cr_{\min}) \cdot \frac{k-1}{|R|-1}$;
10 **if** *Uniform(0,1)* $\leq cr_k$ **then**
11 Assign $R[k]$ in $\mathbf{x}$ a random value from its value set;
12 **if** S_p == *true* **then**
13 Project $\mathbf{x}$ to nearest point in pruned space (Sec. III-B);
14 Append $\mathbf{x}$ to $[\mathbf{x}]$;
15 // Stage 3: Candidate Selection
16 $QoR_{pred} \leftarrow predict([\mathbf{x}])$;
17 $A \leftarrow acquisition(QoR_{pred})$;
18 $\mathbf{x}^* \leftarrow \arg\max_{\mathbf{x} \in [\mathbf{x}]} A$;
19 $T \leftarrow (P == P_{\text{last}})\ ?\ (T+1) : 0$;
20 $S_p \leftarrow S_p \wedge (T < T_{\max})$;
21 **return** $\mathbf{x}^*$

where $\|\mathbf{\Theta}_i\|$ denotes the number of configurable values of x_i. And the total variance of QoR, $\mathrm{Var}[f(\mathbf{x})]$, is computed by

$$\mathrm{Var}[f(\mathbf{x})] = \sum_{\substack{S \subseteq \{1,\ldots,n\} \\ S \neq \emptyset}} \mathbb{E}_{\mathbf{x}_S}\left[f_S(\mathbf{x}_S)^2\right]. \tag{6}$$

Herein, S denotes any subset of input variables, where its size ranges from 1 to n (the total number of directives), $\mathbb{E}_{\mathbf{x}_S}[\cdot]$ denotes the expectation with respect to the distribution of the variable vector $\mathbf{x}_S$. Following this, we extract the importance of each directive, I_i, using Eq. 4, and subsequently rank all the directives based on their quantified importance.

Ranking-based sampling. We then propose a customized sampling strategy leveraging the importance ranking. Firstly, we keep updating the current Pareto frontier with the explored design points. Subsequently, to identify a promising design point for sampling, we fix the entire directive set to an instance in the current Pareto frontier, aiming at conducting exploration in the vicinity of a currently optimal region. Next, a *change rate* is assigned to each inserted directive based on its importance ranking: a directive with higher importance is assigned a lower probability of change, with the change rate increasing as importance decreases. Specifically, the change rates are linearly distributed across the ranked directive list, ranging from a minimum change rate ($cr_{\min}$) to fully random (100%). In this way, we extend the current Pareto frontier by fixing important directives to their configurations on the frontier with higher probabilities, while granting the remaining directives greater flexibility for exploration. The search engine is guided to balance *exploration* of unseen design spaces with *exploitation* of existing Pareto-optimal solutions to maximize immediate performance gains. On this basis, this sampling flow is repeated N times, each time identifying one design point to add to the candidate set, as shown in lines 6-14 of

TABLE II
BENCHMARK DESIGN SPACE SIZES AND ADRS COMPARISON OF AUTOSHRINK AND BASELINES.

Benchmark	Design Space Size		ADRS (%, Arithmetic Mean ± Standard Deviation over 3 Runs)						
	Original	Pruned (ours)	GA	RL	SA	Sherlock [17]	ScaleHLS [12]	Prospector [9]	AutoShrink (ours)
aes	1.24×10^{12}	2.01×10^{7}	93.12±37.59	110.06±47.67	34.34±13.38	91.51±19.19	47.54±10.84	30.40±6.08	**18.38±6.89**
atax	5.31×10^{6}	7.40×10^{3}	41.69±19.74	40.52±12.91	17.45±7.13	42.02±19.96	22.06±6.53	15.60±7.23	**5.28±1.00**
backprop	2.41×10^{14}	9.75×10^{7}	51.52±18.32	55.33±21.23	23.52±15.11	44.51±19.60	31.31±12.80	17.10±7.48	**4.05±1.95**
bicg	6.37×10^{7}	1.11×10^{4}	52.68±4.43	40.95±17.58	24.78±10.66	53.07±3.25	30.93±9.57	30.84±7.56	**5.32±3.22**
fft	1.08×10^{16}	1.37×10^{10}	74.30±23.63	71.55±16.74	31.31±14.38	58.07±18.12	34.47±4.74	31.67±20.00	**12.36±2.99**
gemm	1.42×10^{7}	3.01×10^{4}	30.77±5.89	33.76±6.73	22.13±12.46	55.01±35.43	23.40±3.55	48.82±41.17	**16.56±12.47**
gesummv	9.95×10^{5}	1.20×10^{3}	29.28±13.09	20.57±9.62	10.55±2.56	24.97±5.54	12.96±6.27	14.33±5.97	**2.95±1.64**
md_knn	2.10×10^{5}	9.00×10^{1}	12.02±2.90	8.54±5.06	9.71±0.85	12.23±2.45	7.08±5.16	1.52±1.01	**0.13±0.08**
mvt	1.91×10^{8}	1.11×10^{4}	47.37±4.96	52.37±25.59	29.24±3.62	59.65±16.32	39.27±11.93	25.52±2.84	**19.62±7.06**
spam_filter	4.29×10^{9}	1.62×10^{6}	142.89±45.96	137.28±42.08	80.13±28.58	116.30±60.65	96.51±10.08	18.83±12.26	**1.63±1.35**
spmv	7.78×10^{3}	3.00×10^{1}	17.19±2.12	10.92±0.57	10.13±0.69	12.52±1.57	10.37±0.97	1.55±0.79	**0.25±0.22**
stencil3d	1.22×10^{10}	7.13×10^{6}	79.90±11.69	75.03±19.19	47.10±26.06	72.53±11.04	41.10±37.01	28.32±4.08	**10.64±3.62**
Geo. Mean	/	/	45.50	41.39	23.45	44.21	26.53	15.40	**3.94**
Arith. Mean	/	/	56.06	54.74	28.37	53.53	33.08	22.04	**8.10**

Algorithm 1.

Candidate selection. In each round of exploration, we collect a large number of design points from the ranking-based sampling stage. These points are then evaluated by the surrogate model within Bayesian optimization, ranked by their acquisition function values derived from predicted QoR, and finally, the top-ranked design point is regarded as the most promising candidate, which will be further verified through HLS. When combined with pruning, each sampling point is projected onto the pruned design space (see Section III-B), ensuring that exploration remains confined to this reduced space. To further enhance the efficacy of sampling, we monitor the QoR in each round of exploration and track the number of consecutive rounds without improvement. If the Pareto frontier remains unchanged for more than T_{max} rounds, we consider it unlikely that better solutions exist within the pruned space, so the exploration is redirected to the complete design space to search for additional optimization opportunities.

V. EXPERIMENTAL RESULTS

A. Experimental Setup

AutoShrink builds on OpenBox [22] and employs Bayesian optimization [23] with a Gaussian process surrogate model (RBF kernel) and the multi-objective max-value entropy search acquisition function [24]. AutoShrink's parameters regarding the sampling engine, cr_{min}, N, and T_{max}, are empirically set as 20%, 5000, and 100, respectively. The complete framework is implemented on a workstation with dual Intel Xeon Gold 6230, 251 GiB memory, and Ubuntu 20.04, targeting AMD/Xilinx ZCU102 FPGA board with Vitis HLS 2022.2.

We conduct DSE experiments on 12 representative benchmark applications from PolyBench [25], MachSuite [26] and Rosetta [27], covering 8 distinct domains such as linear algebra, molecular dynamics, and neural networks. QoR metrics are normalized to $[0, 1]$ using min–max scaling for each benchmark. We construct design spaces from a broad range of HLS directives, including (1) function-level directives: inline, dataflow, and loop merge; (2) loop-level directives: pipeline, unroll, and flatten; and (3) array-level directive: array partition. These directives define benchmark design spaces ranging from 7.78×10^{3} to 1.08×10^{16}, as shown in Table II. For each benchmark, the DSE process begins with 100 randomly sampled design points, followed by 24 hours of AutoShrink exploration with online HLS execution. To avoid excessive runtime on any single design point, a 30-minute timeout is imposed on each HLS run. To mitigate randomness, three independent runs are conducted for each benchmark.

We compare AutoShrink with six representative DSE methods. These include three generic optimization methods: genetic algorithm (GA) [28], simulated annealing (SA) [29], and reinforcement learning (RL) [30], as well as three customized DSE methods as SOTA: Sherlock [17], Prospector [9], and ScaleHLS's exploration engine [12]. Note that Prospector can be regarded as the backbone structure of AutoShrink without pruning and importance-guided sampling. To ensure fairness, all methods use the same experimental settings, including initialization method, runtime budget, and timeout configuration.

To evaluate the quality of DSE results, we choose ADRS, as defined in Eq. 1. A smaller ADRS indicates better quality. However, as benchmarks' design spaces reach up to 1.08×10^{16}, it is infeasible to traverse the whole design space to retrieve the exact Pareto frontier for each benchmark. We thus construct a reference Pareto frontier per benchmark by aggregating all samples across all evaluated DSE methods.

B. DSE Performance and Comparison

We conduct DSE experiments with AutoShrink and the above methods to evaluate their ability to discover high-quality Pareto frontiers. As shown in Table II, AutoShrink delivers 5.73×/9.34× improvement in ADRS on arithmetic/geometric means over generic optimization methods (GA, RL, SA), and 4.47×/7.29× improvement over SOTA approaches (ScaleHLS, Sherlock, Prospector), confirming its effectiveness in large design spaces.

These six methods under comparison can be divided into two categories. The first category emphasizes the enhancement of surrogate models, with Sherlock and Prospector as representatives, showing arithmetic mean ADRS of 53.53% and 22.04%, much worse than AutoShrink's 8.10%. These methods are more likely to waste sampling budget on re-

dundant or low-quality design points due to the lack of pruning. As a result, their surrogate models are trained on uninformative data, which undermines prediction accuracy and in turn reduces their ability to identify promising design points. The second category, including GA, RL, SA, and ScaleHLS, focuses on improving search engines, which yield arithmetic mean ADRS between 28.37% and 56.06%, all worse than AutoShrink. Their search engines do not consider the varying impact of directives on QoR and simply perform uniform adjustment on all directives when searching for new design points. This practice hinders fine-grained tuning of crucial directives and ultimately reduces exploration efficacy. Overall, AutoShrink incorporates both comprehensive pruning and importance-guided sampling, effectively addressing the limitations of the aforementioned methods.

C. Ablation Study

We conduct an ablation study to separately evaluate the effect of comprehensive pruning (CP) and importance-guided sampling (IGS) proposed in AutoShrink. Three experiments are conducted, including AutoShrink w/o IGS, AutoShrink w/o CP, and AutoShrink w/o CP&IGS, which are compared with the complete AutoShrink. Herein, 'w/o' denotes 'without'.

Results are shown in Table III. Compared with no optimization (w/o CP&IGS), employing IGS alone (w/o CP) achieves 1.04× and 1.01× improvement on the arithmetic and geometric mean ADRS, respectively, while adding CP alone (w/o IGS) yields the gains of 1.22× and 1.49×. Results indicate that both CP and IGS provide measurable benefits. Furthermore, as demonstrated by the complete AutoShrink, the combination of these two components delivers significantly greater benefits, achieving 2.72× and 3.91× improvement in the arithmetic and geometric mean ADRS, respectively. This compound enhancement can be attributed to the tightly coupled workflow between CP and IGS. CP removes 2–6 orders of magnitude of redundant and inefficient regions, yielding high-quality design spaces. Subsequently, within these condensed spaces, IGS stands a higher chance of identifying promising design points that exactly extend the current Pareto frontier. Hence, these two components, CP and IGS, form a positive feedback loop that justifies a synergistic "$1 + 1 > 2$" effect.

D. Trade-off between Exploration and Exploitation

As for importance sampling, AutoShrink introduces a parameter called minimum change rate (cr_{min}), which controls the trade-off between *exploring unseen designs* and *exploiting the known Pareto frontier*. A larger cr_{min} reduces the degree of exploitation and increases the emphasis on exploration. We investigate how cr_{min} impacts the effectiveness of sampling.

We demonstrate the results of tuning cr_{min} for IGS alone, using `fft` that has the largest design space as a representative case, as shown in Fig. 3. Similar trends are observed across the other benchmarks. Specifically, ADRS improves as cr_{min} increases from 0%, rapidly reaching the best performance at 20%, and subsequently deteriorates with further increases. This implies that, in addition to concentrating the search near the Pareto frontier, adding moderate random variations helps escape local optima by probing unevaluated regions, thereby striking a desirable balance between exploration and exploitation. Furthermore, the optimality observed at a cr_{min} of 20% confirms the effectiveness of importance-guided strategy: the decisions associated with important directives are maintained with an 80% probability, providing dominant control relative to the influence of randomness.

TABLE III
ABLATION STUDY OF AUTOSHRINK RESULTS.
(ADRS %, ARITHMETIC MEAN ± STD OVER 3 RUNS)

Benchmark	Variants for Ablation			AutoShrink
	w/o CP&IGS	w/o CP	w/o IGS	
aes	30.40±6.08	40.86±3.85	23.35±7.20	**18.38±6.89**
atax	15.60±7.23	12.78±7.69	8.68±2.90	**5.28±1.00**
backprop	17.10±7.48	22.91±6.90	6.05±1.20	**4.05±1.95**
bicg	30.84±7.56	26.48±7.76	10.81±1.82	**5.32±3.22**
fft	31.67±20.00	31.20±8.69	71.54±27.44	**12.36±2.99**
gemm	48.82±41.17	25.36±1.16	22.87±12.49	**16.56±12.47**
gesummv	14.33±5.97	13.44±8.90	3.42±1.43	**2.95±1.64**
md_knn	1.52±1.01	3.90±4.37	1.19±0.62	**0.13±0.08**
mvt	25.52±2.84	27.89±2.97	31.79±1.06	**19.62±7.06**
spam_filter	18.83±12.26	22.16±11.58	11.79±6.56	**1.63±1.35**
spmv	1.55±0.79	0.73±0.35	1.97±0.54	**0.25±0.22**
stencil3d	28.32±4.08	25.68±16.00	22.51±14.25	**10.64±3.62**
Geo. Mean	15.40	15.20	10.32	**3.94**
Arith. Mean	22.04	21.11	18.00	**8.10**

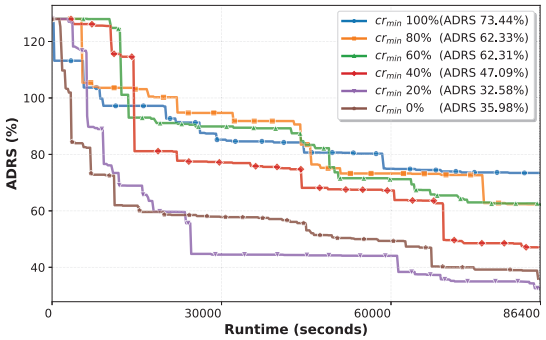

Fig. 3. The curves of ADRS for `fft` during a 24-hour (86400s) DSE process using the AutoShrink variant with only importance-guided sampling, evaluated under different minimum change rates (cr_{min}).

VI. CONCLUSION

This paper proposes AutoShrink, an HLS DSE framework for large-scale design spaces. It has two key components: (1) a comprehensive pruning strategy, which generalizes pruning rules to reduce design points; and (2) an importance-guided sampling approach, which extends the Pareto frontier by leveraging directive importance. Experiments show that AutoShrink outperforms various existing methods, offering high efficiency and strong generalization ability.

ACKNOWLEDGMENT

This work is supported by the Guangdong Basic and Applied Basic Research Foundation (No. 2024A1515013155, 2023A1515110769), the Shenzhen Science and Technology Program (Grant No. RCBS20231211090600003), the National Natural Science Foundation of China (No. 62404257, 62404021), the Beijing Natural Science Foundation (No. 4244107), and the Fundamental Research Funds for the Beijing University of Posts and Telecommunications (No. 2025AI4S20).

REFERENCES

[1] P. Coussy and A. Morawiec, *High-Level Synthesis: From Algorithm to Digital Circuit*. Springer, 2008.

[2] S. Lahti, P. Sjövall, J. Vanne, and T. D. Hämäläinen, "Are we there yet? a study on the state of high-level synthesis," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, vol. 38, no. 5, pp. 898–911, 2018.

[3] AMD/Xilinx Ltd, "Vitis high-level synthesis user guide," White Paper, UG1399 (v2022.2), 2022.

[4] A. Canis, J. Choi, M. Aldham, V. Zhang, A. Kammoona, J. H. Anderson, S. Brown, and T. Czajkowski, "Legup: high-level synthesis for fpga-based processor/accelerator systems," in *The 19th ACM/SIGDA international symposium on Field programmable gate arrays (FPGA)*, 2011, pp. 33–36.

[5] F. Ferrandi, V. G. Castellana, S. Curzel, P. Fezzardi, M. Fiorito, M. Lattuada, M. Minutoli, C. Pilato, and A. Tumeo, "Invited: Bambu: an open-source research framework for the high-level synthesis of complex applications," in *ACM/IEEE Design Automation Conference (DAC)*, 2021, pp. 1327–1330.

[6] B. C. Schafer and Z. Wang, "High-level synthesis design space exploration: Past, present, and future," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, vol. 39, no. 10, pp. 2628–2639, 2020.

[7] F. Ferrandi, P. L. Lanzi, D. Loiacono, C. Pilato, and D. Sciuto, "A multi-objective genetic algorithm for design space exploration in high-level synthesis," in *IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, 2008, pp. 417–422.

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

[9] A. Mehrabi, A. Manocha, B. C. Lee, and D. J. Sorin, "Prospector: Synthesizing efficient accelerators via statistical learning," in *Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2020, pp. 151–156.

[10] L. Ferretti, G. Ansaloni, and L. Pozzi, "Lattice-traversing design space exploration for high level synthesis," in *International Conference on Computer Design (ICCD)*, 2018, pp. 210–217.

[11] Z. Zou, C. Tang, L. Gong, C. Wang, and X. Zhou, "Flexwalker: An efficient multi-objective design space exploration framework for hls design," in *International Conference on Field-Programmable Logic and Applications (FPL)*, 2024, pp. 126–132.

[12] H. Ye, C. Hao, J. Cheng, H. Jeong, J. Huang, S. Neuendorffer, and D. Chen, "Scalehls: A new scalable high-level synthesis framework on multi-level intermediate representation," in *IEEE international symposium on high-performance computer architecture (HPCA)*, 2022, pp. 741–755.

[13] A. Ferikoglou, A. Kakolyris, V. Kypriotis, D. Masouros, D. Soudris, and S. Xydis, "Data-driven hls optimization for reconfigurable accelerators," in *ACM/IEEE Design Automation Conference (DAC)*, 2024, pp. 1–6.

[14] G. Zhong, A. Prakash, S. Wang, Y. Liang, T. Mitra, and S. Niar, "Design space exploration of fpga-based accelerators with multi-level parallelism," in *Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2017, pp. 1141–1146.

[15] A. Sohrabizadeh, Y. Bai, Y. Sun, and J. Cong, "Automated accelerator optimization aided by graph neural networks," in *ACM/IEEE Design Automation Conference (DAC)*, 2022, pp. 55–60.

[16] N. Wu, Y. Xie, and C. Hao, "Ironman-pro: Multiobjective design space exploration in hls via reinforcement learning and graph neural network-based modeling," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, vol. 42, no. 3, pp. 900–913, 2022.

[17] Q. Gautier, A. Althoff, C. L. Crutchfield, and R. Kastner, "Sherlock: A multi-objective design space exploration framework," *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, vol. 27, no. 4, pp. 1–20, 2022.

[18] H. Kuang, X. Cao, J. Li, and L. Wang, "Hgbo-dse: Hierarchical gnn and bayesian optimization based hls design space exploration," in *International Conference on Field Programmable Technology (ICFPT)*, 2023, pp. 106–114.

[19] Q. Sun, T. Chen, S. Liu, J. Chen, H. Yu, and B. Yu, "Correlated multi-objective multi-fidelity optimization for hls directives design," *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, vol. 27, no. 4, pp. 1–27, 2022.

[20] A. B. Kahn, "Topological sorting of large networks," *Communications of the ACM*, vol. 5, no. 11, pp. 558–562, 1962.

[21] F. Hutter, H. Hoos, and K. Leyton-Brown, "An efficient approach for assessing hyperparameter importance," in *International conference on machine learning (ICML)*. PMLR, 2014, pp. 754–762.

[22] Y. Li, Y. Shen, W. Zhang, Y. Chen, H. Jiang, M. Liu, J. Jiang, J. Gao, W. Wu, Z. Yang *et al.*, "Openbox: A generalized black-box optimization service," in *The 27th ACM SIGKDD conference on knowledge discovery & data mining*, 2021, pp. 3209–3219.

[23] B. Shahriari, K. Swersky, Z. Wang, R. P. Adams, and N. De Freitas, "Taking the human out of the loop: A review of bayesian optimization," *Proceedings of the IEEE*, vol. 104, no. 1, pp. 148–175, 2015.

[24] S. Belakaria, A. Deshwal, and J. R. Doppa, "Max-value entropy search for multi-objective bayesian optimization," *Advances in neural information processing systems*, vol. 32, 2019.

[25] L.-N. Pouchet., "Polybench: The polyhedral benchmark suite," http://www.cs.ucla.edu/~pouchet/software/polybench, 2012.

[26] B. Reagen, R. Adolf, Y. S. Shao, G.-Y. Wei, and D. Brooks, "Machsuite: Benchmarks for accelerator design and customized architectures," in *IEEE International Symposium on Workload Characterization (IISWC)*, 2014, pp. 110–119.

[27] Y. Zhou, U. Gupta, S. Dai, R. Zhao, N. Srivastava, H. Jin, J. Featherston, Y.-H. Lai, G. Liu, G. A. Velasquez, W. Wang, and Z. Zhang, "Rosetta: A realistic high-level synthesis benchmark suite for software programmable fpgas," in *ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (FPGA)*, 2018, p. 269–278.

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

[29] P. J. Van Laarhoven and E. H. Aarts, *Simulated annealing*. Springer, 1987.

[30] B. Jang, M. Kim, G. Harerimana, and J. W. Kim, "Q-learning algorithms: A comprehensive classification and applications," *IEEE Access*, vol. 7, pp. 133 653–133 667, 2019.