

A Collaborative Framework for Multi-Level Multi-Objective Design Space Exploration

Jiangnan Li^{1†}, Kaixiang Zhu^{1†}, Yuping Bai¹, Yunfei Dai¹, Qing He², Yu He³ and Lingli Wang^{1*}

¹State Key Laboratory of Integrated Chips and Systems, Fudan University, Shanghai, China

²Department of Electronic Science and Technology, Tongji University, Shanghai, China

³Hangzhou Phlexing Technology Co., Ltd, Hangzhou, China

{jnli22, kxzhu23, ypbai22, yfdai24}@m.fudan.edu.cn, qhe@tongji.edu.cn, yhe@phlexing.com, {luk, llwang}@fudan.edu.cn

Abstract—High-level synthesis (HLS) tools have drawn considerable attention in recent years because they can automatically generate hardware description code from high-level semantics under compiler-controlled configurations. However, the time-consuming design process, the inherent trade-offs among design objectives, and the often suboptimal quality of RTL produced by HLS have meant that prior studies rarely scale to or investigate the downstream stages beyond HLS.

In this paper, we present COLA, an end to end design space exploration (DSE) framework that effectively automates the adaptive tuning of compiler transformation sequences and logic synthesis directives. First, we introduce MOEBO, a holistic Bayesian optimization method that builds multiple local surrogate models within trust regions while maintaining a global surrogate to correct local search bias and align decisions across regions. We further design a cooperative acquisition maximization scheme that coordinates these surrogates to propose diverse and promising candidates in parallel. Additionally, we employ reinforcement learning (RL) techniques to optimize logic synthesis by exploring the design space more effectively, improving the quality of the generated RTL and minimizing the area-delay product. The RL model dynamically adapts the logic synthesis directives to achieve better optimization outcomes over traditional methods. Experimental results show that, our framework achieves a substantial speedup across diverse accelerators for varying kernel granularities with a better trade-off between area and performance.

Index Terms—Design space exploration, Bayesian optimizer, Reinforcement learning

I. INTRODUCTION

The popularity of High-Level Synthesis (HLS) has increased with compilation frameworks supporting diverse high-level inputs such as C/C++, PyTorch, TensorFlow, and ONNX [1]–[5]. Through pipeline optimization, HLS tools can automatically generate register-transfer level (RTL) designs, but different optimization configurations result in significant variations in power, latency, and resource usage. Existing industrial and academic HLS tools [6]–[8] rely on high-level intermediate representations such as LLVM IR [9] for scheduling and analysis, with performance estimated analytically or via downstream synthesis; however, this approach fails to capture additional optimizations performed by logic synthesis tools [10], [11], leading to large discrepancies between early estimates and final quality of results [12].

[†]J. Li and K. Zhu contributed equally to this work.

This work is supported by the National Key R&D Program of China (2021ZD0114701).

In practice, high level synthesis (HLS) and logic synthesis are tightly coupled stages, with their performance metrics exhibiting highly nonlinear dependencies [13]. HLS directives such as loop unrolling, tiling, and pipelining fundamentally shape the structure and complexity of the generated RTL, while logic synthesis applies gate level optimizations that directly influence area, delay, and power. However, the RTL produced by HLS [8] often contains synthesis unfriendly patterns, such as deeply nested control logic, high fanout broadcasts, and complex interconnect structures, which can degrade timing closure and constrain the achievable frequency [14]. For instance, Guo et al. report that broadcast patterns alone can reduce the maximum frequency of synthesized designs by more than 50% [15].

However, multiple optimization flows pursue multiple, often competing, design objectives that are intricately correlated, which poses substantial challenges for prior methods. Existing approaches typically optimize these levels independently, either: focusing on HLS-level design space exploration (DSE) only, or tuning logic synthesis strategies alone. Without incorporating feedback from the synthesis stage, HLS tools lack the critical information needed to avoid the inefficiencies described above. In addition, the performance across the two stages as a function of directive configurations is highly nonlinear, which makes the space difficult to explore with a single method.

To address these challenges, it is necessary to adopt a collaborative design space exploration approach that seamlessly integrates high-level algorithmic frameworks with low-level logic optimization within a unified synthesis flow, thereby improving both design productivity and hardware quality simultaneously. This formulation reveals the strong interdependencies between abstraction levels, which are often ignored in existing single-stage optimization approaches. Moreover, building on ONNX-MLIR [4], we implement a converter from the *ONNX* dialect to the *Linalg* dialect [16] to supply standardized test inputs.

The contributions of this paper are summarized as follows:

- We propose COLA¹, a collaborative framework that explores both compiler transformation and synthesis-directive spaces, aligning optimization objectives across layers and enabling cross-layer interactions to improve the hardware QoR.
- We propose MOEBO, which leverages a lightweight multi-task surrogate and instantiates multiple local agents

¹<https://github.com/jiangnan7/COLA>

together with a global coordinator to characterize the design space, enabling parallel, sample-efficient exploration.

- We introduce an RL-based LS, which automatically explores and fine-tunes synthesis directives. This enables adaptive and intelligent DSE to achieve superior performance trade-offs, while effectively reducing the logic levels by 44% without noticeable area overhead.

II. PRELIMINARIES

A. Problem Formulation

Definition 1 (Pareto optimality). For an m -objective minimization problem with objectives f_1, f_2, \dots, f_m , a solution x dominates another solution x' (denoted $x \preceq x'$) if $\forall i \in \{1, \dots, m\} : f_i(x) \leq f_i(x')$ and $\exists j : f_j(x) < f_j(x')$. A solution is Pareto-optimal if it is not dominated by any other solution. The image of all Pareto-optimal solutions in the objective space forms the Pareto frontier.

Definition 2 (ADRS). Given a reference frontier Γ (e.g., the ground-truth or best-known front) and a learned frontier Ω , the Average Distance to Reference Set (ADRS) is

$$\text{ADRS}(\Gamma, \Omega) = \frac{1}{|\Gamma|} \sum_{\gamma \in \Gamma} \min_{\omega \in \Omega} \|\gamma - \omega\|_2 \quad (1)$$

Smaller ADRS indicates a learned frontier closer to the reference.

Problem (HLS design space exploration) For a given HLS directive design space \mathcal{X} , each configuration $\mathbf{x} \in \mathcal{X}$ is represented as a vector of n directive parameters $\mathbf{x} = [x_1, x_2, \dots, x_n]$. The corresponding QoR metrics $\mathbf{y} = [y_1, y_2, \dots, y_m]$ define the objective space \mathcal{Y} . The goal of design space exploration (DSE) is to identify a set of Pareto-optimal configurations $X = \{\mathbf{x} \in \mathcal{X} \mid \nexists \mathbf{x}' \in \mathcal{X}, \mathbf{x}' \preceq \mathbf{x}\}$ with their corresponding objectives $Y = \{f(\mathbf{x}) \mid \mathbf{x} \in X\}$. The optimization problem is formulated as:

$$\mathbf{x}^* = \arg \min_{\mathbf{x} \in \mathcal{X}} f(\mathbf{x}) = [f_1(\mathbf{x}), f_2(\mathbf{x}), \dots, f_m(\mathbf{x})] \quad (2)$$

B. Bayesian Optimization

BO iterates a surrogate and an acquisition over the search space. Instead of a heavy multi-task GP, we adopt a dependency-free multi-task surrogate that captures inter-objective correlation via covariance factorization and performs fast kernel regression.

We adopt a lightweight multi-task surrogate that captures cross-objective structure with minimal overhead. Concretely, we estimate the empirical cross-objective covariance from the observations, use its Cholesky factor to decorrelate targets, and fit each decorrelated component with Nadaraya Watson kernel regression. Predictions are then mapped back to the original objective space to obtain joint means and calibrated marginal uncertainties. For batch selection, we use Monte Carlo Expected Hypervolume Improvement (EHVI) to maximize the expected gain in hypervolume, optionally augmented with a distance-based diversity term to reduce sample crowding and improve Pareto coverage.

C. Reinforcement Learning

RL is a branch of machine learning concerned with how an agent learns to make sequential decisions by interacting with an environment. At each step, the agent observes the current state, takes an action, and receives a reward signal that measures the immediate or long-term quality of that decision. Over time, the agent refines its policy to maximize cumulative rewards, making RL particularly well-suited for problems with sequential dependencies and delayed outcomes.

The process of logic synthesis can be naturally modeled in this framework. Each synthesis command transforms the circuit, changing attributes such as node count, logic depth, power, and area. Every decision yields a new configuration, which can be viewed as a sequence of state transitions. This makes logic synthesis well-suited to formulation as a Markov Decision Process (MDP), where states represent configurations, actions correspond to commands, and rewards reflect gains in performance, area, power, or other objectives.

III. THE PROPOSED FRAMEWORK

As illustrated in Fig. 1, the proposed framework, COLA, provides a collaborative design-space exploration environment that bridges compiler level optimizations and logic synthesis. COLA first converts the ONNX kernel into a linear algebra representation and applies preliminary optimizations, including memory layout refinement and tensor optimizations. It then composes, for each kernel, configurable compiler transformations such as loop tiling, unrolling, and interchange, emits the corresponding configuration file, and encodes each configuration into an embedding space to support design space exploration. These are passed through an HLS tool, Yosys and FPGA implementation to obtain preliminary RTL implementations and quality of result (QoR) metrics. To efficiently navigate the vast design space, a space modeling module based on trust region modeling and global ensemble strategies is employed, which guides sampling via collaborative acquisition maximization and adapts regions iteratively according to hypervolume improvements. In parallel, the generated RTL is further processed by Yosys and evaluated in a logic synthesis environment powered by ABC, where RL agents interact with the synthesis tool. The RL agent, consisting of actor critic networks, learns from state reward feedback derived from circuit features and synthesis results, thereby optimizing logic level decisions. By jointly aligning compiler transformations and synthesis directives, COLA enables cross layer optimization and collaborative search, leading to significant improvements in hardware QoR across performance, area, and delay metrics.

A. Ensemble Optimization Algorithm.

BO is traditionally driven by a single global surrogate model combined with a sequential acquisition strategy, which often leads to excessive exploration and limited parallelism. Inspired by TuRBO, we localize modeling within multiple trust regions (TRs) and propose *MOEBO* for MLIR-based HLS DSE. *MOEBO*: (a) treats DSE as multi-objective and tracks progress by hypervolume (HV); (b) uses a lightweight multi-task surrogate via covariance factorization plus kernel regression to

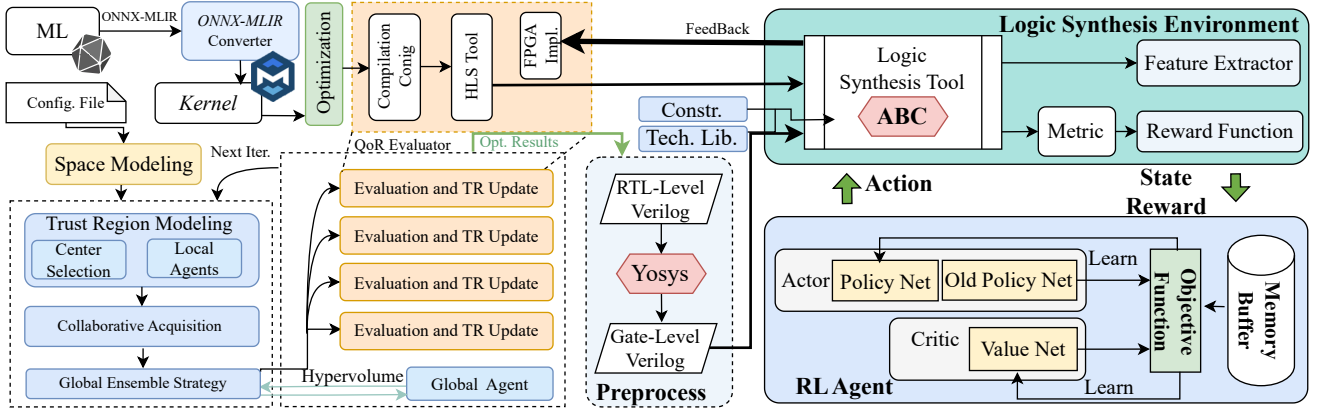


Fig. 1. The proposed collaborative framework.

capture inter-objective structure efficiently; (c) couples K local TR agents with a global agent to balance exploitation and exploration; and (d) selects batches by Monte-Carlo EHVI with a distance-based diversity regularizer, improving the front while avoiding point crowding.

1) *Trust-Region Modeling*: We organize local search with multiple decoupled trust regions (TRs) in the normalized directive space $[0, 1]^d$. Each TR T_i is an axis-aligned hypercube

$$T_i = \{z \in [0, 1]^d : |z_k - z_{\text{center},i,k}| \leq L_i/2, \forall k \in \{1, \dots, d\}\} \quad (3)$$

with edge length $L_i \in [L_{\min}, L_{\max}]$ and center $z_{\text{center},i}$. At the end of the initialization phase, all TR centers are set to the best observed design under a scalarization consistent with “maximize all”.

At iteration t , the surrogate for TR i is fit using only samples that fall inside T_i ; if none exist, it falls back to the global dataset for stability. After proposing and evaluating a batch from one arm, we compute the hypervolume (HV) gain ΔHV and update a success counter s_i or a failure counter f_i accordingly.

This expands promising neighborhoods and shrinks unproductive ones, while recentring contracted TRs to track the most competitive region discovered so far.

2) *Diversity-Aware Acquisition*: Let $\mu(z) \in \mathbb{R}^3$ and $\sigma(z) \in \mathbb{R}_+^3$ denote the predictive means and marginal standard deviations from the multi-task surrogate at candidate z . We score candidates by a Monte Carlo approximation to EHVI augmented with a repulsive diversity term. Specifically, draw S i.i.d. samples

$$y^{(s)} \sim \mathcal{N}(\mu(z), \text{diag}(\sigma^2(z))), \quad s = 1, \dots, S. \quad (4)$$

and estimate

$$\widehat{\text{EHVI}}(z) = \frac{1}{S} \sum_{s=1}^S (\text{HV}(\mathcal{P} \cup \{y^{(s)}\}) - \text{HV}(\mathcal{P})). \quad (5)$$

To discourage clustering around previously sampled points, we penalize the inverse nearest-neighbor distance in the embedded space. For each selected arm, a large pool of candidates is drawn (uniformly within the TR hypercube or globally), deduplicated at the decoded-configuration level, scored, and the top- B points (batch size) are greedily chosen for evaluation. This

rule balances quality (EHVI) and spread (penalty), yielding batches that both improve the front and cover it more uniformly.

3) *Global Ensemble Strategy*: Alongside the N_{TR} local models, we maintain a global surrogate trained on all accumulated samples. At each iteration, a windowed Upper-Confidence-Bound (UCB) multi-armed bandit selects which arm (one of the TRs or the global arm) proposes the next batch. Let \bar{r}_i be the empirical mean HV gain achieved by arm i over a recent window, n_i its number of pulls, and t the total number of pulls so far. The selected arm is

$$i^* = \arg \max_i \bar{r}_i + c \sqrt{\frac{\log t}{n_i + \delta}}, \quad (6)$$

When the global arm is chosen, candidates are sampled across $[0, 1]^d$ and scored by the global surrogate with the same diversity-aware EHVI rule.

First, the global model aggregates information across regions, enabling long jumps to unexplored but promising areas when local TRs stagnate. Second, the bandit converts *observed* HV gains into an adaptive allocation policy that automatically emphasizes productive arms and deprioritizes those with low recent payoff, avoiding hand-tuned schedules. In practice, this yields a robust balance between exploitation within matured TRs and exploration via the global arm, improving sample efficiency under tight evaluation budgets.

B. RL-guided Logic Synthesis

Following the high-level DSE output, we further enhance the design’s performance using an RL-guided logic synthesis framework, as illustrated in Fig. 1. In this framework, we model the synthesis sequence optimization task as an MDP comprising a state space, an action space, a transition function, and a reward function. The state space represents different circuit configurations, where each state encapsulates characteristics such as the number of nodes, logic depth, area, and timing of the design. The transition function determines how the system moves between states based on actions, while the reward function evaluates the hardware design quality after each action. An RL agent interacts with this environment, using feedback from the objective function to learn optimal synthesis

strategies. The agent’s goal is to iteratively improve the circuit’s performance after technology mapping through actions that optimize delay, resource usage, and other design constraints.

While prior RL methods, such as DRiLLS [17], optimize for logic-level proxy metrics like And-Inverter Graph (AIG) node count and depth, these estimates often fail to align with the actual optimization goals of technology mapping. Recognizing this gap, our approach uses actual post-mapping QoR metrics (area and critical path delay) to define the agent’s observations and reward signal. This ensures that the agent directly learns synthesis strategies that improve the final QoR.

Yosys [10] first processes the generated RTL for elaboration and simplification, then passes it to ABC [11] for logic-level optimization and technology mapping. The LS environment, composed of the ABC optimization flow, maintains the current netlist and applies transformations selected by the agent. Each episode corresponds to a full synthesis session on a given design, during which the agent incrementally applies a sequence of synthesis passes.

State: The state observed by the agent is derived from post-mapping QoR metrics, including area, critical path delay, and a one-hot encoding of the most recently applied synthesis pass. These features capture both the quality of the current netlist and the recent synthesis context, enabling the agent to make informed decisions without requiring low-level structural details of the circuit.

Action Space: The action space plays a crucial role in the effectiveness of RL-guided logic synthesis, as it directly defines the set of transformations the agent can sequence and combine. In synthesis sequence optimization, the design of the action space significantly impact both learning efficiency and the quality of final results.

To this end, we construct the action space using the logic optimization passes available in ABC, one of the most widely used and academically recognized open-source synthesis frameworks. ABC provides a rich set of highly optimized, hand-crafted transformations. We treat each synthesis pass as a discrete action, allowing the agent to compose complex strategies through sequential decision-making.

The action space in our framework includes three categories of synthesis operations, expressed as:

$$A = \left\{ \begin{array}{l} \text{b, f, rw, rw -z, rw -l, rw -z -l, rf, rf -z, rf -l,} \\ \text{rf -z -l, rs, rs -z, dc2, \&dsdb, mapD, ifK6, sopb,} \\ \text{blut} \end{array} \right\} \quad (7)$$

where `sopb`, `&dsdb`, and `blut` correspond to SOP-based balancing, DSD-based balancing, and AIG-based balancing operations, respectively. These passes target structural improvements at the logic level to ease subsequent mapping.

The action `mapD` is a compound operation designed for standard-cell-based technology mapping under delay constraints. It internally invokes a sequence of commands: `ifraig`; `map -D Dc`; `mfs2`; `topo`; and `strash`. This sequence maps the AIG network to standard cells with a target delay constraint `Dc`, applies delay-driven optimization using `mfs2`, and converts the result back into AIG form. Incorporating delay constraints during iterative mapping helps guide the RL agent toward delay-optimized solutions. Another compound

action `ifK6` performs LUT-based optimization through the command sequence `&if -K 6 -a`; `&strash`; `&if -K 6 -g`, where the first phase prioritizes area minimization (`a`) and the second focuses on delay optimization (`g`). These commands allow the agent to explore synthesis strategies targeting both ASIC (standard-cell) and FPGA (LUT-based) design styles within a unified framework.

Reward function: At the end of each episode, the agent receives a numerical reward based on the final mapped design quality. The reward is defined as:

$$r_t = 1 - \left(\frac{Area_t}{Area_{ini}} \right)^\alpha \cdot \left(\frac{Delay_t}{Delay_{ini}} \right)^\beta, \quad (8)$$

where α and β are user-defined weights balancing area and delay optimization. The reward is normalized by the initial design’s area ($Area_{ini}$) and delay ($Delay_{ini}$). This normalization is crucial as it makes the reward signal independent of the absolute metric values, which can vary significantly across different circuits. This allows the agent to learn a more generalizable optimization policy and stabilizes the training process. This reward function encourages the discovery of synthesis sequences that yield tangible improvements in real hardware QoR.

Agent: We adopt the RL agent implementation from the open-source RL Zoo 3 [18], built on top of Stable-Baselines3 [19]. In our experiments, we use the Proximal Policy Optimization (PPO) algorithm due to its stability and efficiency in high-dimensional, discrete action spaces. The agent is trained in an environment that simulates the Yosys+ABC synthesis flow, where each episode corresponds to a complete synthesis session on a design. For each target RTL design, a dedicated agent is trained by interacting with the synthesis environment. Through this process, the agent learns a specialized policy to identify the optimal sequence of synthesis commands that minimizes area or delay. To ensure stable and effective training, we employ reward normalization and entropy regularization, which helps the agent to thoroughly explore the solution space and find superior synthesis strategies for the specific circuit.

IV. EXPERIMENTAL EVALUATION

To evaluate our proposed compiler framework, we conduct a series of experiments targeting FPGA implementations. We focus on key hardware metrics, including clock cycles, area, power consumption, and overall design quality after synthesis. In our experiments, we generate application-specific accelerators and synthesize them using our proposed logic synthesis optimization framework, which integrates Yosys [10] and ABC [11] for frontend logic synthesis and technology mapping. Then, we implement the designs using VTR [20] flow on an academic FPGA architecture based on Stratix IV: `k6_frac_N10_frac_chain_mem32K_40nm`.

A. Comparison with existing frameworks

For baseline comparisons, we consider traditional high level synthesis flows using Bambu 2024.10 [8] (with default optimization settings enabled) and SODA [3]. We evaluate our method on representative kernels from machine learning

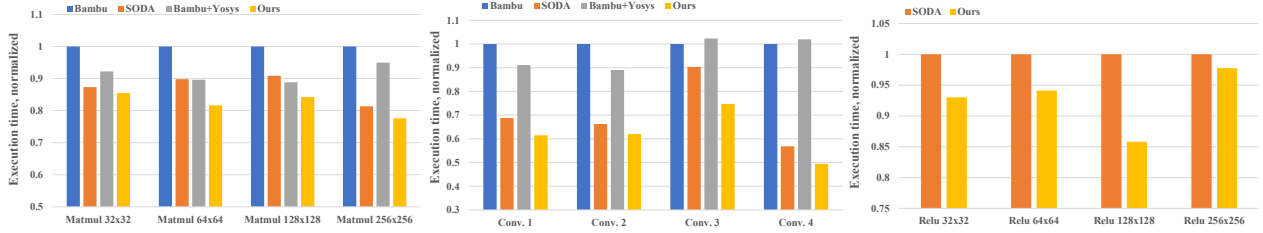


Fig. 2. Normalized execution time across different benchmarks.

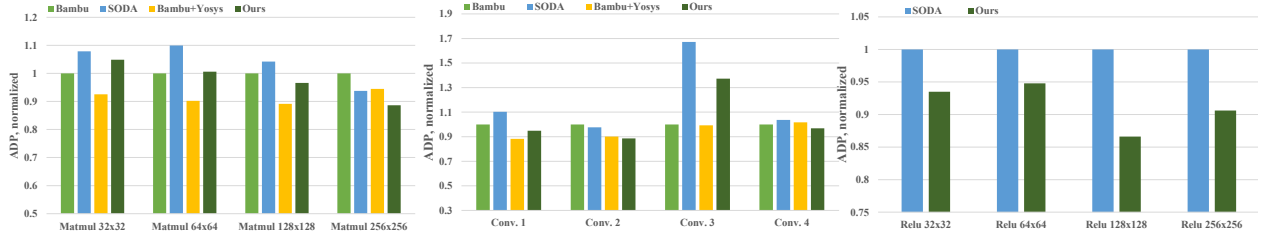


Fig. 3. Normalized area delay product across different benchmarks.

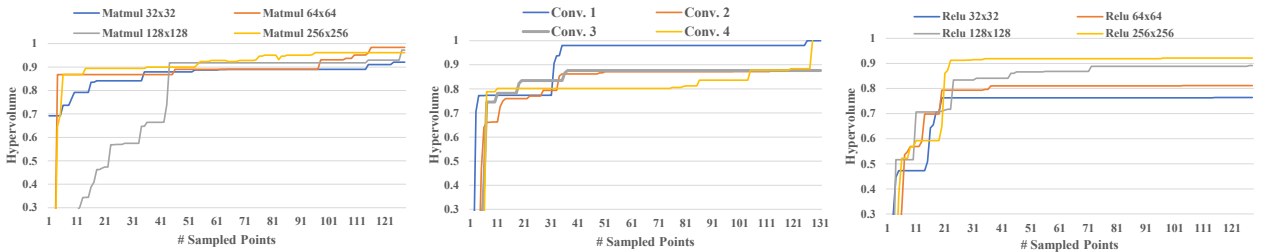


Fig. 4. Hypervolume trends across different cases.

workloads. For matmul and ReLU, we use square tensors of sizes 32×32 to 256×256 , covering both small and large scales. For convolution, we consider MLIR implementations with kernel sizes 3×3 to 7×7 and input/output channels drawn from $\{1, 8, 16\}$, with spatial resolutions ranging from 32×32 to 64×64 . We refer to the evaluated convolution cases as Conv. 1 to Conv. 4. Our approach demonstrates the potential advantages of tight integration between HLS and logic synthesis, highlighting improvements in both execution time (defined as the product of cycle count and delay) and area delay product (ADP), where ADP is measured as the product of area and delay. Both area and delay are obtained from VTR reports, with delay reported in *ns*.

Figures 2 and 3 show that Ours consistently improves both execution time and area delay product (ADP) across matrix multiplication, convolution, and activation workloads. Using the figure normalization, Ours reduces execution time by roughly 5% to 18% on matmul kernels, and by about 9% to 38% on several convolution cases. On activation operators, Ours delivers steady gains of about 2% to 7%. Compared to SODA, the ADP results mirror these trends: typical reductions are about 6% to 15% on matmul and activation tasks, with larger benefits on selected convolution cases. For ReLU we present SODA and Ours. ReLU is element wise and memory bound; once

vectorization and tiling are fixed, the normalized trends are consistent across flows.

A major source of delay inaccuracy in HLS lies in how functional units are ultimately realized during logic synthesis. The RTL produced by HLS typically specifies operations at an abstract level (for example, “+” and “*”), whereas logic synthesis tools implement them using concrete micro-architectures such as ripple carry adders (RCA), carry save adders (CSA), or carry lookahead adders (CLA), and structured multiplier trees. These architectural choices, together with technology mapping, gate sizing, and retiming, determine the true critical path of the final netlist. Because such downstream optimizations are outside the HLS cost model, pre-synthesis delay estimates often diverge from post-synthesis measurements.

The quality of the Pareto frontier is measured by ADRS and hypervolume. Figure 4 visualizes the evolution of hypervolume as the number of sampled points increases across matmul, conv, and reLU cases, showing consistent convergence behavior.

B. LS result

The logic optimization result of the proposed framework is illustrated in Table I, which compares the performance of three synthesis methods: the initial design (Original), resyn2 from ABC, and our proposed method.

TABLE I
LOGIC OPTIMIZATION RESULTS COMPARED WITH ABC RESYN2.

Design		origin			ABC resyn2				Ours			
		#Node	#Lev.	Quality	#Node	#Lev.	Quality	Impr.(%)	#Node	#Lev.	Quality	Impr.(%)
Vanilla	Matmul 32x32	14383	76	1093108	9350	73	682550	38	10263	39	400257	63
	Matmul 64x64	14317	76	1088092	9285	73	677805	38	10556	37	390572	64
	Matmul 128x128	14281	76	1085356	9264	73	676272	38	10395	37	384615	65
	Matmul 256x256	14189	76	1078364	9216	73	672768	38	10461	38	397518	63
	Conv. 1	17459	76	1326884	10711	71	760481	43	11750	37	434750	67
	Conv. 2	29696	62	1841152	16972	59	1001348	46	18322	43	787846	57
	Conv. 3	27401	62	1698862	16123	60	967380	43	16899	43	726657	57
	Conv. 4	27072	62	1678464	15844	60	950640	43	16666	42	699972	58
	Relu 32x32	3040	34	103360	1546	34	52564	49	1959	18	35262	66
	Relu 64x64	3016	34	102544	1546	34	52564	49	1927	19	36613	64
	Relu 128x128	3003	32	96096	1537	32	49184	49	1848	19	35112	63
	Relu 256x256	2983	33	98439	1566	34	53244	46	1889	18	34002	65
Optimized	Matmul 32x32	14556	76	1106256	9438	73	688974	38	10675	38	405650	63
	Matmul 64x64	14488	76	1101088	9374	73	684302	38	10727	37	396899	64
	Matmul 128x128	14499	76	1101924	9369	73	683937	38	10467	37	387279	65
	Matmul 256x256	14405	76	1094780	9327	73	680871	38	10315	38	391970	64
	Conv. 1	17854	76	1356904	10719	71	761049	44	12072	37	446664	67
	Conv. 2	29708	48	1425984	16973	47	797731	44	17736	36	638496	55
	Conv. 3	29501	60	1770060	16608	57	946656	47	18121	44	797324	55
	Conv. 4	26931	60	1615860	15535	58	901030	44	17013	43	731559	55
	Relu 32x32	3019	30	1816452	1545	30	46356	97	1858	18	33088	78
	Relu 64x64	2975	34	159895	1562	35	54710	66	1884	19	34947	99
	Relu 128x128	3009	32	7428421	1951	31	59968	99	2207	16	35308	100
	Relu 256x256	2970	32	7257866	1922	31	59065	99	2199	16	34261	100
Geo. mean		10285	54	1066458	6554	54	353173	48	6889	30	206583	66
Ratio		1.00	1.00	1.00	0.59	0.97	0.35	-	0.67	0.56	0.23	-

In our experiments, the focus of our optimization is on delay and α is set to 0 and β is set to 1 in Eq. (8). To evaluate the effectiveness of our method, we use the **Quality** parameter:

$$\text{Quality} = \#Node * \#Lev. \quad (9)$$

As shown in Table I, our method achieves a substantial improvement in overall performance compared to `resyn2`. Specifically, it reduces the logic levels ($\#Lev.$) by 44%, while keeping the area overhead to only 8%, and at the same time delivers an average 18% improvement in the overall Quality parameter. This shows that our framework not only achieves effective delay optimization, but also provides balanced area–delay trade-offs, resulting in faster and more efficient circuits.

V. RELATED WORK

HLS DSE has progressed from analytical modeling to machine learning based methods. Many studies employ active learning [21], [22] or Bayesian optimization [13], [23] to search for Pareto optimal designs; for example, HGBO-DSE [23] combines enhanced Latin hypercube sampling with weight decay in MOTPE [24] to accelerate convergence. Yet BO methods often assume continuous parameters and rely on a single global surrogate, leading to over exploration and suboptimal results [23], [25]. A complementary line of work strengthens the compiler substrate to make exploration more tractable. ScaleHLS [1] applies loop optimizations with MLIR [26], [27], and AutoScaleDSE [28] reduces the search space with a random forest classifier. However, most pipelines still lower high level descriptions to C++ for FPGA synthesis [6] rather than passing LLVM IR directly to HLS tools [2], [8]. The RTL generated by HLS [8] often contains structures difficult for mid end and

back end tools to optimize [14], [15], and back end synthesis is usually treated as fixed without feedback to the front end, causing avoidable performance loss.

Deep reinforcement learning frameworks like DRiLLS [17] and AISYN [29] train agents to explore synthesis transformations directly on post-HLS netlists, achieving significant improvements in area and timing compared to heuristic-based flows. Other methods, such as Q-ALS [30] use Q-learning to guide technology mapping under error and timing constraints. GNN-based approaches (e.g., ABC-RL) employ graph representations of logic networks to inform RL decisions over AIG operations [31], further enhancing synthesis quality. However, these methods assume that input RTL is already well structured for synthesis, without addressing upstream schedule-induced structural inefficiencies. Our work bridges this gap by integrating Bayesian multi-objective optimization for MLIR-based compilation with RL-guided logic synthesis, enabling coordinated optimization across abstraction levels and delivering better end-to-end hardware quality.

VI. CONCLUSION

In this paper, we propose a collaborative framework that integrates HLS and LS, aligning their optimization objectives and enabling synergistic interactions that significantly impact the final hardware design. COLA thus forms a complementary and goal-coherent flow, in which structured MLIR transformations provide backend-friendly RTL and RL-guided logic synthesis adapts to the characteristics of upstream schedules. Together, they enable efficient and synthesizable accelerators with improved delay-performance trade-offs.

REFERENCES

- [1] 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.
- [2] J. Li, Z. Zhang, X. Zhou, and L. Wang, "An MLIR-Based Compiler for Hardware Acceleration with Recursion Support: (PhD Forum Paper)," in *2024 International Conference on Field Programmable Technology (ICFPT)*, 2024, pp. 1–4.
- [3] N. Bohm Agostini, S. Curzel, V. Amaty, C. Tan, M. Minutoli, V. G. Castellana, J. Manzano, D. Kaeli, and A. Tumeo, "An MLIR-Based Compiler Flow for System-Level Design and Hardware Acceleration," in *Proceedings of the 41st IEEE/ACM International Conference on Computer-Aided Design*, ser. ICCAD '22. New York, NY, USA: Association for Computing Machinery, 2022.
- [4] T. Jin, G.-T. Bercea, T. D. Le, T. Chen, G. Su, H. Imai, Y. Negishi, A. Leu, K. O'Brien, K. Kawachiya, and A. E. Eichenberger, "Compiling ONNX Neural Network Models Using MLIR," 2020. [Online]. Available: <https://arxiv.org/abs/2008.08272>
- [5] J. Li, X. Cao, K. Zhu, W. Yin, and L. Wang, "Dynvec: An end-to-end framework for efficient vector-dataflow execution," in *2025 IEEE/ACM International Conference On Computer Aided Design (ICCAD)*, 2025, pp. 1–9.
- [6] X. Inc, "Vitis high-level synthesis user guide (ug1399)," 2023.
- [7] Google Open Source, "XLS: Accelerated hardware development," <https://github.com/google/xls>, 2025.
- [8] F. Ferrandi, V. G. Castellana, S. Curzel, P. Fezzardi, M. Fiorito, M. Latuada, M. Minutoli, C. Pilato, and A. Tumeo, "Invited: Bambu: an open-source research framework for the high-level synthesis of complex applications," in *2021 58th ACM/IEEE Design Automation Conference (DAC)*, 2021, pp. 1327–1330.
- [9] C. Lattner and V. Adve, "LLVM: a compilation framework for lifelong program analysis transformation," in *International Symposium on Code Generation and Optimization, CGO.*, 2004, pp. 75–86.
- [10] C. Wolf, "Yosys open synthesis suite," <https://yosyshq.net/yosys/>.
- [11] R. Brayton and A. Mishchenko, "Abc: An academic industrial-strength verification tool," in *Computer Aided Verification*, T. Touili, B. Cook, and P. Jackson, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 24–40.
- [12] H. Ye, D. Z. Pan, C. Leary, D. Chen, and X. Xu, "Subgraph Extraction-Based Feedback-Guided Iterative Scheduling for HLS," in *2024 Design, Automation Test in Europe Conference Exhibition (DATE)*, 2024, pp. 1–6.
- [13] Q. Sun, T. Chen, S. Liu, J. Chen, H. Yu, and B. Yu, "Correlated Multi-objective Multi-fidelity Optimization for HLS Directives Design," *ACM Trans. Des. Autom. Electron. Syst.*, vol. 27, no. 4, Mar. 2022. [Online]. Available: <https://doi.org/10.1145/3503540>
- [14] M. Tatsuoka, R. Watanabe, T. Otsuka, T. Hasegawa, Q. Zhu, R. Okamura, X. Li, and T. Takabatake, "Physically aware high level synthesis design flow," in *2015 52nd ACM/EDAC/IEEE Design Automation Conference (DAC)*, 2015, pp. 1–6.
- [15] L. Guo, J. Lau, Y. Chi, J. Wang, C. H. Yu, Z. Chen, Z. Zhang, and J. Cong, "Analysis and optimization of the implicit broadcasts in fpga hls to improve maximum frequency," in *2020 57th ACM/IEEE Design Automation Conference (DAC)*, 2020, pp. 1–6.
- [16] "Linalg dialect." [Online]. Available: <https://mlir.llvm.org/docs/Dialects/Linalg/>
- [17] A. Hosny, S. Hashemi, M. Shalan, and S. Reda, "Drills: Deep reinforcement learning for logic synthesis," in *2020 25th Asia and South Pacific Design Automation Conference (ASP-DAC)*, 2020, pp. 581–586.
- [18] A. Raffin, "RL baselines3 zoo," <https://github.com/DLR-RM/rl-baselines3-zoo>, 2020.
- [19] A. Raffin, A. Hill, A. Gleave, A. Kanervisto, M. Ernestus, and N. Dormann, "Stable-baselines3: Reliable reinforcement learning implementations," *Journal of Machine Learning Research*, vol. 22, no. 268, pp. 1–8, 2021. [Online]. Available: <http://jmlr.org/papers/v22/20-1364.html>
- [20] M. A. Elgammal, A. Mohaghegh, S. G. Shahrouz, F. Mahmoudi, F. Koşar, K. Talaie, J. Fife, D. Khadivi, K. Murray, A. Boutros *et al.*, "VTR 9: Open-Source CAD for Fabric and Beyond FPGA Architecture Exploration," *ACM Transactions on Reconfigurable Technology and Systems*, 2024.
- [21] H.-Y. Liu and L. P. Carloni, "On learning-based methods for design-space exploration with High-Level Synthesis," in *2013 50th ACM/EDAC/IEEE Design Automation Conference (DAC)*, 2013, pp. 1–7.
- [22] P. Meng, A. Althoff, Q. Gautier, and R. Kastner, "Adaptive Threshold Non-Pareto Elimination: Re-thinking machine learning for system level design space exploration on FPGAs," in *2016 Design, Automation Test in Europe Conference Exhibition (DATE)*, 2016, pp. 918–923.
- [23] H. Kuang, X. Cao, J. Li, and L. Wang, "HGBO-DSE: Hierarchical GNN and Bayesian Optimization based HLS Design Space Exploration," in *2023 International Conference on Field Programmable Technology (ICFPT)*, 2023, pp. 106–114.
- [24] Y. Ozaki, Y. Tanigaki, S. Watanabe, M. Nomura, and M. Onishi, "Multiobjective Tree-Structured Parzen Estimator," *J. Artif. Int. Res.*, vol. 73, May 2022. [Online]. Available: <https://doi.org/10.1613/jair.1.13188>
- [25] J. Snoek, H. Larochelle, and R. P. Adams, "Practical Bayesian optimization of machine learning algorithms," in *Proceedings of the 26th International Conference on Neural Information Processing Systems - Volume 2*, ser. NIPS'12. Red Hook, NY, USA: Curran Associates Inc., 2012, p. 2951–2959.
- [26] "Affine dialect." [Online]. Available: <https://mlir.llvm.org/docs/Dialects/Affine/>
- [27] "Circuit ir compilers and tools." [Online]. Available: <https://circuit.llvm.org/>
- [28] H. Jun, H. Ye, H. Jeong, and D. Chen, "AutoScaleDSE: A Scalable Design Space Exploration Engine for High-Level Synthesis," *ACM Trans. Reconfigurable Technol. Syst.*, vol. 16, no. 3, Jun. 2023. [Online]. Available: <https://doi.org/10.1145/3572959>
- [29] G. Pasandi, S. Pratty, and J. Forsyth, "Aisyn: Ai-driven reinforcement learning-based logic synthesis framework," 2023. [Online]. Available: <https://arxiv.org/abs/2302.06415>
- [30] G. Pasandi, S. Nazarian, and M. Pedram, "Approximate logic synthesis: A reinforcement learning-based technology mapping approach," in *20th International Symposium on Quality Electronic Design (ISQED)*, 2019, pp. 26–32.
- [31] K. Zhu, M. Liu, H. Chen, Z. Zhao, and D. Z. Pan, "Exploring logic optimizations with reinforcement learning and graph convolutional network," in *2020 ACM/IEEE 2nd Workshop on Machine Learning for CAD (MLCAD)*, 2020, pp. 145–150.