DynaOpt: A Heterogeneous Logic Optimization Framework with Dynamic Sequence Generation

Xingyu Qin, Guande Dong, Jianwang Zhai†, Kang Zhao
Beijing University of Posts and Telecommunications
{qinxingyu, 2025010575, zhaijw, zhaokang}@bupt.edu.cn

***Abstract*—Heterogeneous logic optimization improves circuit quality by partitioning a design and leveraging the best Directed Acyclic Graph (DAG) representation for each region. However, existing frameworks are limited by their reliance on applying fixed, pre-defined optimization scripts to these partitions. This approach fails to adapt to the specific structure of each partition or its impact on global circuit metrics. This paper introduces DynaOpt, a framework that overcomes this limitation by dynamically generating tailored optimization sequences. After partitioning the circuit with a timing and structure-aware algorithm and selecting the optimal DAG for each partition, DynaOpt discovers a bespoke optimization sequence for each sub-circuit. The key to this process is our novel, globally-aware fitness function, which guides a Genetic Algorithm (GA) by efficiently approximating the impact of local changes on the final circuit quality. Experiments demonstrate that DynaOpt achieves a significant improvement in Quality of Results (QoR) over the state-of-the-art (SOTA) framework. This validates the effectiveness of generating custom optimization sequences and addresses the fundamental limitations of relying on pre-defined sequences.**

I. Introduction

Logic synthesis is a foundational step in Electronic Design Automation (EDA), transforming high-level circuit descriptions into optimized gate-level netlists that determine the final circuit's Power, Performance, and Area (PPA). Logic optimization applies a sequence of transformations to a circuit's Boolean network [1], but a central challenge is that any fixed optimization sequence struggles to unlock the full potential of diverse circuits, often leading to suboptimal local optima [2].

To address this, two significant research frontiers have advanced in parallel. The first focuses on dynamic sequence generation, where methods learn or search for a superior, circuit-specific optimization flow. Techniques range from data-driven modeling [3], [4] and Reinforcement Learning (RL) [5]–[8] to online search strategies like Monte Carlo Tree Search (MCTS) [9]. These powerful methods treat circuits monolithically, applying a globally-determined sequence and overlooking structural diversity in large designs [10].

The second frontier is heterogeneous logic optimization, which acknowledges that different parts of a circuit benefit from different underlying data structures (e.g., AIGs [11], MIGs [12], XAGs [13], XMGs [13]). Pioneering frameworks like LSOracle [14] and HeLO [10] partition a circuit and select the best-fit representation for each part, demonstrating significant PPA improvements. However, after this sophisticated partitioning, these frameworks revert to applying a simple, pre-defined, and fixed optimization recipe to each partition.

†Corresponding author.

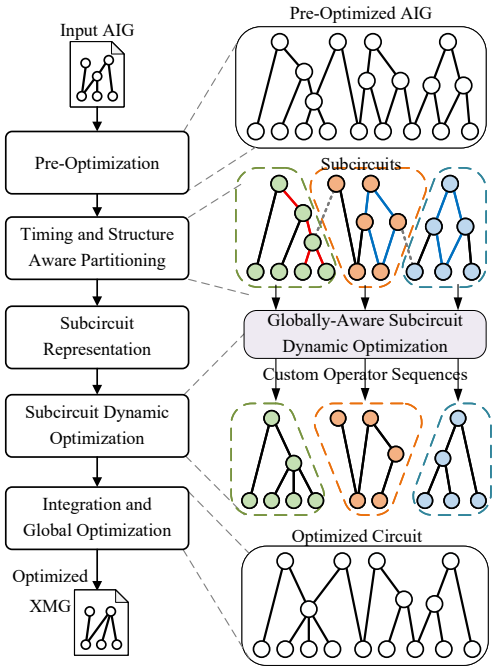

Fig. 1 Overall flow of DynaOpt framework.

A critical disconnect thus exists: one field has mastered the "how" of optimization (dynamic sequences) but is blind to the "where" (internal heterogeneity), while the other has mastered the "where" but is limited in the "how". This leaves a crucial optimization dimension unexplored and motivates our central question: How can we unify the spatial intelligence of heterogeneous partitioning with the procedural intelligence of dynamic sequence generation?

This paper introduces DynaOpt, a novel framework that directly answers this question by bridging the gap between these two frontiers. DynaOpt pioneers a methodology that is both heterogeneous in representation and adaptive in its optimization process. The contributions are:

- We introduce a global pre-optimization method that generates an initial synthesis sequence, performing preliminary circuit optimization to prepare for the subsequent, more complex heterogeneous optimization phase.
- We employ a timing and structure-aware partitioning algorithm designed to preserve local structures with high optimization potential by tuning the weights of graph edges according to their timing and structural importance.
- We design a Graph Neural Network (GNN)-based model to analyze each partition and select its most effective DAG representation.

978-3-9826741-1-7/DATE26/© 2026 EDAA

- We introduce a globally-aware fitness function to guide a GA in dynamically generating an optimization sequence for each partition. This approach replaces the pre-defined recipes of prior work by evaluating the true global impact of local optimizations.

By crafting a bespoke optimization strategy for each part of the circuit, DynaOpt achieves a new level of granularity. This dynamic approach outperforms the SOTA HeLO [10], underscoring the effectiveness of integrating heterogeneous representations with dynamic optimization sequences.

II. Preliminaries

A. Logic Synthesis and Logic Optimization

Logic synthesis is the automated process of converting a high-level hardware description into a mapped gate-level netlist. A critical phase in this flow is technology-independent optimization. Here, the circuit is first represented as a generic data structure, such as a DAG. A dominant representation is the And-Inverter Graph (AIG), a homogeneous DAG composed of two-input AND gates and inverters [15].

Within this technology-independent stage, logic optimization aims to restructure the DAG to improve the final circuit's PPA. This is achieved by applying a sequence of transformations. To balance the often-conflicting goals of minimizing area (node count) and delay (logic depth) before technology mapping, the Node-Depth Product (NDP) is a widely adopted metric to guide the optimization algorithms.

B. Heterogeneous Logic Optimization

Traditional optimization uses a single data structure (e.g., AIG) for the entire design. A more advanced approach, heterogeneous logic optimization, partitions the circuit and applies the most suitable representation—such as MIGs, XAGs, or XMGs—to each region [10], unlocking powerful optimization operators unique to each data structure. This enables a more tailored, spatially-aware optimization.

The pioneering work, LSOracle [14], introduced this concept by partitioning a circuit, using a DNN to predict the best data structure for each partition, and then applying a fixed optimization script. Building on this, HeLO [10] employed a more sophisticated hierarchical clustering for partitioning and a GNN for prediction. By creating more coherent partitions and using a more powerful model, HeLO established a new SOTA, but it still relied on predetermined, static optimization scripts for each logic representation.

C. Circuit and Hypergraph Partitioning

Effective heterogeneous optimization hinges on high-quality circuit partitioning. A hypergraph naturally models this, where a hyperedge connects multiple vertices, representing a net connecting multiple gates. The goal of k-way Hypergraph Partitioning (HGP) is to divide vertices into k balanced disjoint blocks while minimizing a cost function on cut hyperedges. A standard objective is minimizing the connectivity metric $\sum_{e \in E_{\text{cut}}} (\lambda(e) - 1) w(e)$, where $\lambda(e)$ denotes the number of partitions spanned by hyperedge e and $w(e)$ is its weight.

As the HGP problem is NP-hard, heuristic-based multilevel partitioners like KaHyPar [16] are widely used. Crucially, these tools allow assigning custom weights to hyperedges. By increasing an edge's weight, the heuristic is penalized more for cutting it, making it more likely that the connected vertices remain in the same block. This weighting mechanism provides a powerful handle to create more structurally and functionally aware partitions, moving beyond simple topology-driven cuts.

D. Problem Formulation

We formulate the heterogeneous logic optimization problem as finding a strategy $\mathcal{S}$ that transforms an initial logic network C into a functionally equivalent network C' with the best possible QoR. For technology-independent optimization, this QoR is typically the NDP.

This strategy $\mathcal{S}$ is a composite of 3 interdependent decisions:

1) **Partitioning ($\mathcal{P}$):** Dividing the global circuit C into a set of k disjoint sub-circuits $\{c_1, \ldots, c_k\}$ that collectively reconstruct C. The goal is to group structurally and functionally related logic.
2) **DAG Representation ($\mathcal{R}$):** Assigning each sub-circuit c_i to its most suitable DAG representation r_i from a set of available options, such as $\{$AIG, MIG, XAG, XMG$\}$.
3) **Optimization Sequence ($\mathcal{O}$):** For each sub-circuit c_i in its assigned representation r_i, generating a bespoke sequence of optimization operators $O_i = (o_1, \ldots, o_L)$.

The overall objective is to discover the optimal strategy $\mathcal{S}^* = (\mathcal{P}^*, \mathcal{R}^*, \mathcal{O}^*)$ that yields the globally optimal circuit C':

$$\mathcal{S}^* = \arg\min_{\mathcal{S}} \text{QoR}(\text{Apply}(\mathcal{S}, C)). \quad (1)$$

The central challenge lies in the immense and deeply coupled search space. The optimal partitioning ($\mathcal{P}$) is unknown without knowing the best optimization for each partition, while the effectiveness of an optimization sequence ($\mathcal{O}_i$) depends on its local structure, its assigned representation ($\mathcal{R}_i$), and its ultimate impact on the global QoR. Our work, DynaOpt, is designed to navigate this complex interplay by cohesively solving for these components with a global objective in mind.

III. The DynaOpt Framework

To address the limitations of existing heterogeneous logic optimization, we introduce DynaOpt, a framework designed to integrate a timing and structure-aware partitioning strategy with a dynamic, per-partition optimization approach. As illustrated in Fig. 1, DynaOpt implements a multi-stage workflow that progressively refines a logic network by tailoring strategies at both global and local levels.

The process begins with an input AIG, which first undergoes a pre-optimization step using a globally-applied, model-generated sequence. Next, our timing and structure-aware partitioning algorithm divides the network into smaller sub-circuits. For each resulting sub-circuit, a GNN-based classifier predicts the most suitable DAG representation (AIG, MIG, XAG, or XMG), and the partition is converted accordingly. Following this, a GA dynamically discovers a custom optimization sequence for the partition. Critically, the GA's fitness function is globally-aware, evaluating each potential sequence based on its estimated impact on the entire circuit's quality, not just the local partition.

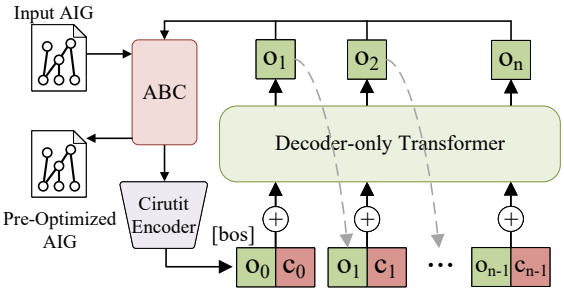

Fig. 2 An illustration of Global Pre-Optimization. The process starts with a [bos] token (o_0). At each step i, the current circuit state is encoded into an embedding c_i, which the model uses to predict the next optimal operator o_{i+1}.

Finally, to preserve the results of the heterogeneous optimization and facilitate a lossless reassembly, all optimized partitions are converted into a unified XMG representation. As a superset of AIG, MIG, and XAG, the XMG format allows for this conversion without any loss of quality [12], [17], [18]. The partitions are then reassembled, and two `rewrite` operations are applied to recover optimization opportunities lost at partition boundaries. This section details each of these core stages.

A. Global Pre-Optimization

Initial partitioning quality is paramount in heterogeneous optimization. However, raw logic networks often contain superficial redundancies that obscure their true structure, leading to suboptimal partitioning. To address this, we introduce a pre-optimization stage to simplify logic and produce a netlist that better reflects the circuit's architectural properties.

We adopt a learning-based approach, inspired by GPT-LS [4], to automatically generate pre-optimization sequences. Our decoder-only Transformer model is trained offline using a dataset of random, short optimization sequences—each composed of operators from {`rw, rwz, rs, rsz, rf, rfz, b`}—applied to diverse benchmark circuits. Only sequences that demonstrably improve final QoR are retained, ensuring the model learns strategies beneficial for our flow. The model uses a dual-input circuit encoder (GNN for topology, MLP for scalar features) as shown in Fig. 4, co-trained with the sub-circuit classification task (Section III-C).

At inference time, for a new circuit, the trained model autoregressively predicts and applies a tailored optimization sequence (Fig. 2), iteratively encoding the circuit state and updating the netlist. The resulting circuit is then input to the main optimization stages.

B. Timing and Structure-Aware Partitioning

The efficacy of heterogeneous optimization hinges on the quality of the initial circuit partitioning. However, prior strategies often lack the necessary awareness to produce optimal partitions. For instance, while methods like LSOracle [14] allow for varying the number of partitions, their underlying cut-driven approach is not inherently adaptive to the circuit's structure. To investigate this, we analyzed LSOracle's performance on the EPFL benchmark suite [19] across different partition counts ($k = 2$ to 10). As shown in Section III-A, the optimal

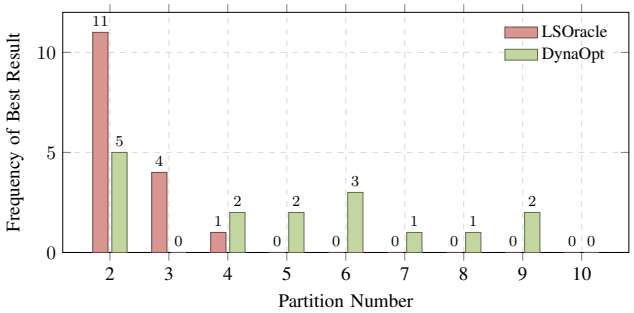

Fig. 3 Best result distribution over partition numbers on EPFL benchmarks.

results for LSOracle are overwhelmingly concentrated at low partition counts ($k = 2$ and $k = 3$). This suggests that simply increasing the partition count does not guarantee better QoR, as the partitioning strategy itself may not effectively utilize the finer granularity. This inflexibility can sever timing-critical paths or break apart structurally significant regions, leading to suboptimal synthesis outcomes.

To address this limitation, we introduce a flexible and extensible timing and structure-aware partitioning scheme. Instead of treating all nets as equal, we assign weights to hyperedges based on their structural and temporal importance. Our approach utilizes standard hypergraph partitioning machinery, which we configure to minimize the Connectivity metric. This objective is chosen because it directly corresponds to minimizing the number of newly created Primary Inputs (PIs) and Outputs (POs) across all partitions, a generally desirable outcome for synthesis. By biasing the partitioner with our novel, domain-specific weighting function, we preserve critical structures within single partitions and maximize their optimization potential. The weight of each net is calculated based on the following considerations.

1) Timing-Awareness

The timing performance of a circuit is fundamentally determined by its critical path. While the logic depth of the pre-mapping critical path is not a direct proxy for post-mapping timing, minimizing it remains a crucial and widely accepted heuristic for achieving better final performance. To preserve the integrity of this path during partitioning (e.g., the red path in Fig. 1), we first identify it and significantly increase the weights of all nets lying on it. To create a "protective buffer", this weight increase is propagated outwards by two hops from the path, with the effect decaying with distance. This strategy encourages the partitioner to keep critical path segments intact within a single sub-circuit, where their logic depth can be more effectively optimized.

2) Structure-Awareness

We enhance weights based on two key structural features.

Reconvergent Paths: These structures, where a signal from a "stem" node fans out and later reconverges (e.g., the blue edges in Fig. 1), are prime locations for logic restructuring [20]. We identify and increase the weights of their nets, limiting the search to a path length of 6 for performance.

High-Fanout Nets: A node with high fanout signifies significant logic reuse. We assign a higher weight to these

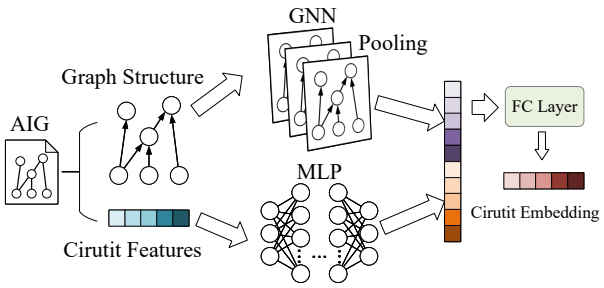

Fig. 4 The architecture of our circuit encoder.

nets to maintain the integrity of these highly shared signals.

The final weight of a hyperedge is the average of its constituent net weights. By providing the partitioner with these carefully crafted weights, our method achieves a sophisticated balance: it minimizes inter-partition connections while preserving crucial timing and structural network characteristics, enabling more effective downstream optimization.

C. Sub-Circuit Representation

After partitioning, we must select the most suitable DAG representation for each sub-circuit to maximize its optimization potential. To automate this decision, we employ a GNN-based classifier that predicts the optimal representation based on the sub-circuit's unique topological and structural features.

Our classification model consists of a circuit encoder and an MLP-based classification head. The circuit encoder (Fig. 4), sharing the pre-optimization architecture (Section III-A), is composed of a 3-layer Graph Attention Network (GAT) and an MLP. The GAT processes the circuit's graph structure, while the MLP processes scalar features (e.g., counts of PIs, POs, and gate types). Their outputs are combined and projected through a final Fully Connected (FC) layer to produce a graph-level embedding fed to the classification head. Crucially, the encoder is trained jointly with this classification task. Its learned weights are frozen and reused as a feature extractor for the pre-optimization model, ensuring a rich, general-purpose representation beneficial for both tasks.

To generate robust training data, we derived ground-truth labels via rigorous empirical evaluation. For a large corpus of sub-circuits, each was optimized multiple times using every candidate DAG representation within our GA-based framework (Section III-D). The representation yielding the best average NDP was assigned as the ground-truth label, ensuring our classifier is trained to make demonstrably effective decisions for our downstream optimization.

D. Globally-Aware Sub-Circuit Dynamic Optimization

After converting each sub-circuit to its optimal DAG representation, we dynamically generate a targeted optimization sequence for it. In this work, we use a GA for this purpose, where each chromosome represents a fixed-length sequence of standard synthesis operators.

The central challenge is defining an effective fitness function. A naive fitness based on a local metric (e.g., the sub-circuit's NDP) is flawed, as local improvements do not guarantee global optimality. While recalculating the global NDP after every local modification would provide a perfect fitness score, its high computational overhead is prohibitive for dynamic methods like GA. To overcome this, we propose a lightweight approximate fitness function that efficiently models the global impact of local changes.

Let N_g and D_g be the node count and depth of the global circuit prior to local optimization. For a given optimization sequence applied to a sub-circuit, let ΔN_l be the resulting reduction in local node count, and let $D_{l,\text{new}}$ be the new local depth. We project the new global node count, N'_g, and depth, D'_g, as follows:

$$N'_g = N_g - \Delta N_l. \tag{2}$$

The calculation of the projected global depth, D'_g, depends on whether the sub-circuit lies on a critical path:

$$D'_g = \begin{cases} D_g - \Delta D_l, & \text{if on crit. path,} \\ \max(D_g, D_{l,\text{new}}) - \lambda \cdot \Delta D_l, & \text{otherwise.} \end{cases} \tag{3}$$

Here, ΔD_l is the reduction in the sub-circuit's depth. When a sub-circuit is not on the critical path, the term $\max(D_g, D_{l,\text{new}})$ penalizes any depth increase that might create a new critical path, while the factor λ (a small constant, e.g., 0.1) provides a soft incentive to reduce depth even on non-critical paths. The fitness of a candidate sequence is then defined as the estimated global NDP reduction:

$$\text{Fitness} = (N_g \cdot D_g) - (N'_g \cdot D'_g). \tag{4}$$

This globally-aware fitness function provides a much tighter correlation to true circuit quality than local metrics, guiding the GA to discover holistic optimization sequences that would be inaccessible to methods reliant on local-only criteria.

IV. Evaluation

A. Implementation and Environment

The GNN and Transformer models are developed with Python using DGL and PyTorch. Pre-optimization is carried out with Yosys [21] and ABC [11]. The DynaOpt framework's heterogeneous optimization is implemented in C++ based on LSOracle [14], an open source logic synthesis framework and Mockturtle [22], a logic network library supporting key representations, including AIGs, MIGs, XAGs, and XMGs. All experiments were conducted on a Linux machine with an Intel(R) Xeon(R) 8383 CPU @ 2.60GHz and 4 NVIDIA RTX 3090 GPUs with CUDA Driver 12.2.

Our models were trained on datasets constructed from 1302 circuits and sub-circuits, which were collected from a wide range of benchmarks, including EPFL [19], ISCAS'85 [23], ISCAS'89 [24], ITC'99 [25], LGSynth'89 [26], LGSynth'91 [27], IWLS'93 [28], IWLS 2005 [29], LEKO/LEKU [30], and OpenCores [31]. The dataset for the sub-circuit classifier was labeled by empirically identifying the best-performing DAG representation for each sub-circuit (Section III-C). For the pre-optimization model, the training data consisted of short synthesis sequences filtered to include only those that demonstrably improved the final QoR (Section III-A). All models were trained using the Adam optimizer.

TABLE I Technology-independent logic optimization result. NDP denotes the product of node count and depth.

Circuit	Original			LSOracle [14]			HeLO [10] *			DynaOpt (Ours)		
	node	level	NDP	node	level	NDP	node	level	NDP	node	level	NDP
pico-rv	14551	31	451081	15124	18	272232	19268	18	346824	12780	15	**191700**
chip_bridge	58596	31	1816476	59180	19	1124420	58317	19	1108023	57532	17	**978044**
s38417	8594	28	240632	9198	19	174762	9522	16	152352	8271	16	**132336**
fpu	65750	33	2169750	60955	20	1219100	68099	20	1361980	60466	18	**1088388**
aes_core	13232	44	582208	14318	29	415222	21867	18	393606	9968	26	**259168**
des_perf	82373	20	1647460	79784	16	1276544	70176	15	**1052640**	76482	14	1070748
ethernet	67164	33	2216412	64909	22	1427998	71896	20	1437920	61065	20	**1221300**
dyn_node	3986	27	107622	4113	19	78147	4034	18	72612	3798	14	**53172**
vga_lcd	105334	22	2317348	102031	19	1938589	101534	17	1726078	103788	15	**1556820**
fpga_bridge	318081	41	13041321	312498	31	9687438	324356	24	7784544	322419	20	**6448380**
i2c	1342	20	26840	1404	10	14040	1385	8	11080	1168	8	**9344**
mem_ctrl	46836	114	5339304	52819	69	3644511	56592	61	3452112	46258	54	**2497932**
normalize	1.000	1.000	1.000	1.018	0.669	0.680	1.106	0.591	0.640	0.929	0.551	**0.511**

TABLE II ASIC technology mapping result using ASAP7 PDK. ADP denotes the product of delay (ps) and area (um^2).

Circuit	Original			LSOracle [14]			HeLO [10] *			DynaOpt (Ours)		
	area	delay	ADP	area	delay	ADP	area	delay	ADP	area	delay	ADP
pico-rv	700.7	325.2	227850.4	618.6	269.5	166728.6	831.6	290.0	241153.0	564.0	283.8	**160051.9**
chip_bridge	2657.2	308.9	820871.5	2613.4	293.6	**767247.8**	3028.2	263.9	799010.0	2512.1	317.1	796448.6
s38417	418.7	314.7	131790.0	428.9	296.7	127280.7	416.2	266.7	**111016.0**	420.6	296.2	124578.0
fpu	2751.4	396.5	1090926.1	2765.4	341.0	**943098.4**	3127.9	324.8	1016209.0	2873.2	336.0	965449.3
aes_core	573.1	511.8	293308.1	615.6	441.7	271895.5	1061.0	251.0	266276.0	562.0	451.9	**253954.4**
des_perf	4459.3	264.8	1180943.2	4492.9	239.1	1074117.6	3457.8	232.8	**804807.0**	4210.7	249.1	1049009.2
ethernet	3131.6	345.1	1080736.1	2872.1	342.4	983464.5	3407.0	289.4	985948.0	2739.1	314.5	**861461.8**
dyn_node	190.1	299.5	56932.0	196.5	241.0	**47353.2**	205.5	231.7	47610.0	199.4	240.9	48026.2
vga_lcd	4766.1	387.1	1844898.0	4676.2	340.2	1591043.9	5627.5	259.5	1460459.2	4292.4	338.8	**1454100.2**
fpga_bridge	14044.7	515.1	7234430.1	14997.7	430.1	6451123.6	16385.7	340.9	**5585048.8**	14458.8	455.7	6588735.1
i2c	53.2	132.1	7024.8	60.3	107.0	6451.3	60.2	131.7	7931.6	57.4	107.4	**6164.6**
mem_ctrl	1826.5	997.9	1822553.1	1935.5	849.0	1643125.7	2395.2	1021.3	2446214.2	1841.7	826.8	**1522807.8**
normalize	1.000	1.000	1.000	1.014	0.877	0.887	1.170	0.812	0.932	0.972	0.886	**0.859**

HeLO* is non-open source, and results are quoted from original paper. The discrepancy primarily arises from Verilog-to-AIG conversion.

B. Experimental Settings

We evaluate our proposed framework, DynaOpt, against two competitive baselines: the open-source LSOracle [14], and the current SOTA work, HeLO [10]. The evaluation is conducted on two distinct benchmark suites.

First, for comparison with the closed-source HeLO, we use the benchmarks from their publication and cite their reported results directly. Since converting Verilog to AIG via `yosys` [21] may introduce initial structural variations, we re-ran LSOracle on our specific netlists. Second, for a more controlled evaluation and our ablation studies, we perform a detailed comparison against LSOracle on the AIG-based EPFL benchmark suite [19].

Following the approach of HeLO, both DynaOpt and LSOracle were run with partition counts ranging from 2 to 10 for each circuit, with the best-performing result reported for each. For its internal parameters, DynaOpt uses a pre-optimization length of $L = 10$ and a final GA-generated sequence of length $L = 10$ (population size 8, 5 generations). To ensure fairness, the set of optimization operators available to DynaOpt is identical to those in LSOracle's fixed scripts.

For technology-independent comparison, all netlists are evaluated on the XMG representation. Both DynaOpt and LSOracle natively produce XMGs. Although HeLO reports results as MIGs, any MIG can be losslessly converted to an XMG [12], [13], ensuring all metrics are directly comparable.

For post-synthesis evaluation, we map all optimized netlists to the ASAP 7nm standard cell library [32] (typical-typical corner, RVT cells). To maintain consistency, we use the exact mapping script provided by LSOracle within ABC. Post-mapping area and delay are extracted via `stime -c`, with the Area-Delay Product (ADP) serving as the primary metric.

C. Comparison with SOTA on HeLO Benchmarks

This section evaluates DynaOpt against LSOracle and the SOTA method, HeLO, on the benchmark suite that established the SOTA, ensuring a direct and relevant comparison.

1) Technology-Independent Results

TABLE I presents the technology-independent results, normalized to the initial circuit's NDP (1.0). DynaOpt demonstrates superior quality, reducing the average NDP to 0.511, significantly surpassing both the normalized HeLO result (0.640) and the LSOracle baseline (0.680). Moreover, DynaOpt improves upon the LSOracle baseline by 24.9% (0.511 vs. 0.680), whereas the original HeLO literature reports only an 8.0% (0.619 vs. 0.673 in [10]) gain over its corresponding baseline. Notably, DynaOpt achieves the best NDP on 11 out of 12 benchmarks, underscoring the consistent power of our dynamic, globally-aware optimization strategy compared to prior static approaches.

2) Technology-Mapping Results

The post-mapping results, shown in TABLE II, confirm that our technology-independent advantages translate effectively to physical metrics. Normalized to the initial ADP (1.0), DynaOpt achieves the lowest average ADP of 0.859. It is noteworthy that the mapping results from the original HeLO [10] appear less competitive than those from the LSOracle baseline

TABLE III In-depth analysis and ablation study on the EPFL benchmark suite.

benchmark	Original			LSOracle [14]			w/o Pre-Opt			w/o Dynamic-Seq			w/o Aware-Part			DynaOpt (Ours)		
	node	level	NDP	node	level	NDP	node	level	NDP	node	level	NDP	node	level	NDP	node	level	NDP
adder	1020	255	260100	1697	21	35637	1459	19	27721	1896	19	36024	1203	18	**21654**	1633	14	22862
cavlc	693	16	11088	707	15	10605	631	11	6941	663	11	7293	608	11	6688	627	10	**6270**
ctrl	174	10	1740	128	7	896	108	5	540	98	5	490	94	5	**470**	95	5	475
div	57247	4372	250283884	121464	841	102151224	74606	1171	87363626	60763	814	49461082	84798	684	58001832	39222	668	**26200296**
i2c	1342	20	26840	1404	10	14040	1356	8	10848	1256	8	10048	1196	7	**8372**	1188	8	9504
int2float	260	16	4160	251	13	3263	236	8	1888	217	10	2170	208	9	1872	220	8	**1760**
log2	32060	444	14234640	38853	281	10917693	32903	160	**5264480**	36692	224	8219008	33697	171	5762187	32635	162	5286870
max	2865	287	822255	4361	70	305270	3525	51	179775	3066	52	159432	4317	27	116559	3770	28	**105560**
mem_ctrl	46836	114	5339304	52819	69	3644511	54642	48	2622816	48246	53	2557038	54891	46	2524986	43060	53	**2282180**
multiplier	27062	274	7414988	37351	139	5191789	29647	88	**2608936**	33240	118	3922320	46445	57	2647365	32014	83	2657162
priority	978	250	244500	1054	130	137020	568	96	54528	624	61	38064	518	45	23310	539	43	**23177**
router	257	54	13878	415	17	7055	205	11	**2255**	242	11	2662	225	11	2475	234	11	2574
sin	5416	225	1218600	6483	141	914103	5865	97	568905	5653	122	689666	6736	91	612976	5399	91	**491309**
sqrt	24618	5058	124517844	17863	5061	90404643	27310	764	20864840	17266	4034	69651044	27482	681	**18715242**	24043	930	22359990
square	18484	250	4621000	22161	50	1108050	14627	38	555826	19020	43	817860	11787	27	**318249**	12986	29	376594
voter	13758	70	963060	10310	76	783560	7584	42	318528	8090	51	412590	4666	39	**181974**	4949	41	202909
normalize	1.000	1.000	1.000	1.210	0.560	0.590	0.976	0.352	0.322	0.977	0.421	0.377	0.997	0.313	0.282	0.900	0.321	**0.266**

in our experiments. This anomaly in relative performance stems from differences in the initial circuit generation and mapping flows within the experimental environments.

While the performance margin narrows after mapping due to the non-linear translation from logic to physical metrics, it is significant that DynaOpt still delivers the best final ADP. This underscores the robustness of the logic structures discovered by our framework in a complete synthesis flow.

D. In-depth Analysis on EPFL Benchmarks

To isolate the sources of our performance gains, we conduct an in-depth analysis of the AIG-based EPFL benchmarks. This controlled environment allows for a direct comparison with LSOracle and a rigorous ablation study quantifying the contribution of each of our architectural innovations.

As summarized in TABLE III, our full DynaOpt framework demonstrates a commanding advantage, achieving an average NDP reduction of 73.4% from the initial circuits. This substantially outperforms LSOracle, which achieves a 41.0% reduction, reaffirming the benefits of our holistic approach. This adaptability is visualized in Section III-A. While LSOracle's optimal performance clusters at small partition counts ($k = 2$ or 3), DynaOpt's is distributed far more evenly. This highlights the limitation of a fixed strategy and confirms the value of our adaptive search over different granularities.

The primary focus of this analysis is the ablation study, which systematically deconstructs our framework to validate each key component. We tested three ablated versions: 1) **`w/o Pre-Opt`**, which disables the global pre-optimization; 2) **`w/o Dynamic-Seq`**, which substitutes our dynamic sequence generation with LSOracle's fixed optimization script; and 3) **`w/o Aware-Part`**, which replaces our partitioning algorithm with the one used by LSOracle. These results are telling. An interesting nuance is that the `w/o Aware-Part` ablation occasionally yields slightly better level improvements. We attribute this to our approximate fitness function, which can mis-prioritize sub-critical paths, causing them to become critical post-optimization. However, the full DynaOpt framework's consistently superior NDP confirms its more effective global trade-off. While all components contribute, the dynamic discovery of a bespoke optimization sequence is the most significant performance driver. This empirically proves DynaOpt's quality stems from its synergistic design, with the globally-aware GA search at its core.

TABLE IV Runtime comparison in seconds (s).

Circuit	LSOracle [14]	DynaOpt (Ours)
adder	6	32
cavlc	7	166
ctrl	4	67
div	634	2094
i2c	6	54
int2float	4	13
log2	314	611
max	12	83
mem_ctrl	340	519
multiplier	189	993
priority	6	76
router	4	24
sin	37	155
sqrt	101	534
square	90	569
voter	54	166
normalize	0.228	1.000

Finally, TABLE IV presents the runtime comparison. On average, LSOracle's runtime is 0.228x that of DynaOpt. This runtime overhead is an expected trade-off from the extensive search space explored by our GA to discover higher-quality solutions. The significant leap in optimization quality justifies this additional computational investment for applications where final circuit performance is paramount.

V. Conclusion

This paper introduces DynaOpt, a heterogeneous logic optimization framework that adaptively generates custom optimization sequences. The proposed approach first leverages a timing and structure-aware partitioning method, followed by a GNN to assign the optimal DAG representation to each partition. Crucially, a GA then generates a bespoke optimization sequence for each sub-circuit, guided by a globally-aware fitness metric. Experimental results demonstrate that DynaOpt outperforms the SOTA framework, establishing a new, more adaptive direction for logic synthesis.

Acknowledgment

This work is supported by the National Natural Science Foundation of China (No. 62404021), the Beijing Natural Science Foundation (No. 4244107), the State Key Lab of Processors, Institute of Computing Technology, CAS (No. CLQ202504), and the Fundamental Research Funds for the Beijing University of Posts and Telecommunications (No. 2025AI4S20).

REFERENCES

[1] C. Yu, H. Xiao, and G. De Micheli, "Developing synthesis flows without human knowledge," in *ACM/IEEE Design Automation Conference (DAC)*, 2018, pp. 1–6.

[2] W. L. Neto, Y. Li, P.-E. Gaillardon, and C. Yu, "FlowTune: End-to-end automatic logic optimization exploration via domain-specific multiarmed bandit," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, vol. 42, no. 6, pp. 1912–1925, 2023.

[3] M. Zhao, J. Liu, J. Zhai, and C. Shi, "MILS: Modality interaction driven learning for logic synthesis," in *ACM Great Lakes Symposium on VLSI (GLSVLSI)*, 2025, pp. 64–70.

[4] C. Lv, Z. Wei, W. Qian, J. Ye, C. Feng, and Z. He, "GPT-LS: Generative pre-trained transformer with offline reinforcement learning for logic synthesis," in *IEEE International Conference on Computer Design (ICCD)*, 2023, pp. 320–326.

[5] A. Hosny, S. Hashemi, M. Shalan, and S. Reda, "DRiLLS: Deep reinforcement learning for logic synthesis," in *IEEE/ACM Asia and South Pacific Design Automation Conference (ASPDAC)*, 2020, pp. 581–586.

[6] K. Zhu, M. Liu, H. Chen, Z. Zhao, and D. Z. Pan, "Exploring logic optimizations with reinforcement learning and graph convolutional network," in *ACM/IEEE Workshop on Machine Learning CAD (MLCAD)*, 2020, pp. 145–150.

[7] G. Dong, J. Zhai, H. Cheng, X. Yang, C. Shi, and K. Zhao, "PIRLLS: Pretraining with imitation and RL finetuning for logic synthesis," in *IEEE/ACM Asia and South Pacific Design Automation Conference (ASPDAC)*, 2025, pp. 65–71.

[8] G. Dong, H. Cheng, J. Zhai, X. Yang, C. Shi, and K. Zhao, "ORL-LO: Offline Reinforcement Learning for Pretraining and Finetuning in Logic Optimization," *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, 2025.

[9] Z. Pei, F. Liu, Z. He, G. Chen, H. Zheng, K. Zhu, and B. Yu, "AlphaSyn: Logic synthesis optimization with efficient monte carlo tree search," in *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, 2023, pp. 1–9.

[10] Y. Pu, F. Liu, Z. He, K. Zhu, R. Fu, Z. Wang, T.-Y. Ho, and B. Yu, "HeLO: A heterogeneous logic optimization framework by hierarchical clustering and graph learning," in *ACM International Symposium on Physical Design (ISPD)*, 2025, pp. 116–124.

[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. Springer, 2010, pp. 24–40.

[12] L. Amarú, P.-E. Gaillardon, and G. De Micheli, "Majority-Inverter Graph: A new paradigm for logic optimization," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, vol. 35, no. 5, pp. 806–819, 2016.

[13] S. Rai, A. T. Calvino, H. Riener, G. De Micheli, and A. Kumar, "Utilizing XMG-based synthesis to preserve self-duality for RFET-based circuits," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, vol. 42, no. 3, pp. 914–927, 2023.

[14] W. L. Neto, M. Austin, S. Temple, L. Amaru, X. Tang, and P.-E. Gaillardon, "LSOracle: a logic synthesis framework driven by artificial intelligence: Invited paper," in *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, 2019, pp. 1–6.

[15] A. Grosnit, M. Zimmer, R. Tutunov, X. Li, L. Chen, F. Yang, M. Yuan, and H. Bou-Ammar, "Lightweight structural choices operator for technology mapping," in *ACM/IEEE Design Automation Conference (DAC)*, 2023, pp. 1–6.

[16] Y. Akhremtsev, T. Heuer, P. Sanders, and S. Schlag, "Engineering a direct k-way hypergraph partitioning algorithm," in *2017 Proceedings of the Ninteenth Workshop on Algorithm Engineering and Experiments (ALENEX)*. SIAM, 2017, pp. 28–42.

[17] I. Háleček, P. Fišer, and J. Schmidt, "Are XORs in logic synthesis really necessary?" in *2017 IEEE 20th International Symposium on Design and Diagnostics of Electronic Circuits Systems (DDECS)*, 2017, pp. 134–139.

[18] W. Haaswijk, M. Soeken, L. Amarù, P.-E. Gaillardon, and G. De Micheli, "A novel basis for logic rewriting," in *IEEE/ACM Asia and South Pacific Design Automation Conference (ASPDAC)*, 2017, pp. 151–156.

[19] L. G. Amarù, P.-E. Gaillardon, and G. D. Micheli, "The EPFL combinational benchmark suite," 2015. [Online]. Available: https://api.semanticscholar.org/CorpusID:13971503

[20] H. Riener, S.-Y. Lee, A. Mishchenko, and G. De Micheli, "Boolean rewriting strikes back: Reconvergence-driven windowing meets resynthesis," in *IEEE/ACM Asia and South Pacific Design Automation Conference (ASPDAC)*, 2022, pp. 395–402.

[21] C. Wolf, J. Glaser, and J. Kepler, "Yosys -a free verilog synthesis suite," *Austrian Workshop on Microelectronics (Austrochip)*, 2013.

[22] M. Soeken, H. Riener, W. Haaswijk, E. Testa, B. Schmitt, G. Meuli, F. Mozafari, S.-Y. Lee, A. Tempia Calvino, and G. Marakkalage, Dewmini Sudara De Micheli, "The EPFL logic synthesis libraries," Jun. 2022, arXiv:1805.05121v3.

[23] F. Brglez and H. Fujiwara, "A neutral netlist of 10 combinational benchmark circuits and a target translator in fortran," in *IEEE International Symposium on Circuits and Systems (ISCAS)*, 1985, pp. 677–692.

[24] F. Brglez, D. Bryan, and K. Kozminski, "Combinational profiles of sequential benchmark circuits," in *IEEE International Symposium on Circuits and Systems (ISCAS)*, May 1989, pp. 1929–1934 vol.3.

[25] F. Corno, M. Reorda, and G. Squillero, "RT-level ITC'99 benchmarks and first ATPG results," *Design Test of Computers, IEEE*, vol. 17, no. 3, pp. 44–53, Jul 2000.

[26] S. Yang, "Logic synthesis and optimization benchmarks," 1989 MCNC International Workshop on Logic Synthesis, Tech. Rep., Dec. 1988.

[27] ——, *Logic synthesis and optimization benchmarks user guide: version 3.0*. Microelectronics Center of North Carolina (MCNC) Research Triangle Park, NC, USA, Jan. 1991.

[28] K. McElvain, "IWLS'93 benchmark set: Version 4.0," a part of IWLS'93 benchmark set, Tech. Rep., May 1993.

[29] C. Albrecht, "IWLS 2005 benchmarks," IWLS, Tech. Rep., Jun. 2005.

[30] J. Cong and K. Minkovich, "Optimality study of logic synthesis for lut-based FPGAs," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, vol. 26, no. 2, pp. 230–239, Feb 2007.

[31] "Opencores," https://opencores.org/.

[32] L. T. Clark, V. Vashishtha, L. Shifren, A. Gujja, S. Sinha, B. Cline, C. Ramamurthy, and G. Yeric, "ASAP7: A 7-nm finfet predictive process design kit," *Microelectronics Journal*, vol. 53, pp. 105–115, 2016.