Performance Pragma-Based Design Space Pruning and Exploration for High-Level Synthesis

Donghao Guo[1], Zhe Lin[2] †, Jianwang Zhai[1], Kang Zhao[1] †
[1]School of Integrated Circuits, Beijing University of Posts and Telecommunications, Beijing, China
[2]School of Integrated Circuits, Sun Yat-sen University, Shenzhen Campus, China
{guodonghao, zhaijw, zhaokang}@bupt.edu.cn, linzh235@mail.sysu.edu.cn

***Abstract*—High-level synthesis (HLS) tools are widely used to design field-programmable gate arrays (FPGAs) and application-specific integrated circuits (ASICs) using high-level languages like C/C++. Existing tools typically provide a set of directives for optimizing generated designs, necessitating exploration of the vast design space formed by directive combinations to obtain satisfactory designs. To tackle this, numerous HLS design space exploration (DSE) frameworks have been proposed in recent years. However, these frameworks either inadequately prune the design space or lack pruning method, leading to excessively long exploration time. This work replaces traditional pragma combinations with the newly released PERFORMANCE pragma in Vitis HLS. Our approach eliminates unreasonable pragma combinations and substantially reduces the total number of pragmas employed. It also prunes pragma parameter values to exclude redundant designs, thereby further shrinking the design space. The proposed framework employs graph neural networks (GNNs) to predict the quality of results (QoR) of candidate designs and adopts Bayesian optimization (BO) for efficient exploration. Experimental results demonstrate that our approach achieves a Pareto front comparable to the state-of-the-art (SOTA) with a 38.30% improvement in ADRS.**

***Keywords*—FPGA, HLS, Design Space Pruning, DSE**

I. INTRODUCTION

Field-programmable gate arrays (FPGAs) [1] are extensively used to implement domain-specific accelerators (DSAs) owing to their flexibility and high energy efficiency [2]. However, traditional FPGA design methodologies rely on hardware description languages (HDLs, e.g., Verilog and VHDL), which require rich hardware expertise and considerable time. High-level synthesis (HLS) tools automatically convert behavioral-level descriptions into register-transfer level (RTL), enabling FPGA development using high-level languages such as C/C++. This greatly lowers the development barrier and significantly improves design efficiency [3].

Existing commercial HLS tools (e.g., Vitis HLS) typically provide users with a set of directives for local optimization (also called pragmas in Vitis HLS). By adjusting the types, positions, and parameters of applied pragmas, users can obtain customized designs. While this offers design freedom, it also poses new challenges: design quality varies significantly across different directive combinations. Users always prioritize designs with optimal quality of results (QoR), which necessitates exploring the vast design space formed by various directive combinations [4].

†Corresponding author

In recent years, numerous HLS design space exploration (DSE) frameworks [5] (e.g., [6], [7], [8], [9], [10], [11], [12]) have been proposed to address this issue. However, the design spaces defined by these frameworks contain many unreasonable pragma combinations. For instance, they overlook that HLS tools automatically unroll inner loops when the PIPELINE pragma is applied to outer loops, leading to excessively long exploration time.

To further lower the development barrier, Vitis HLS offers the PERFORMANCE pragma for loop optimization. This pragma functions as an expert, intelligently assigning low-level pragmas and their parameters based on user-specified parameters. Consequently, the traditional approach of exploring the design space via multiple loop-optimization pragmas has become obsolete: it is not only inefficient but also frequently yields suboptimal or even invalid designs.

Building on this, we replace traditional loop optimization pragma combinations with the PERFORMANCE pragma, substantially shrinking the design space without compromising exploration quality. Our framework employs graph neural networks (GNN) [13] to predict the QoR of candidate designs [14] and leverages Bayesian optimization (BO) for efficient exploration of the design space.

The main contributions are as follows:

- Our framework leverages the newly released PERFORMANCE pragma in Vitis HLS, eliminating numerous unreasonable pragma combinations and substantially reducing the total number of pragmas employed.
- We deduced the algorithmic logic underlying the PERFORMANCE pragma and utilized this insight to further prune the design space.
- By integrating an accurate QoR prediction model with an efficient DSE method, our framework achieves Pareto fronts superior to the state-of-the-art (SOTA) with a lower average ADRS.

II. PRELIMINARIES

A. Design Space Exploration for HLS

HLS tools typically provide users with numerous directives for optimizing generated designs. For example, Vitis HLS 2025.1 offers 29 pragmas. However, most pragmas lack fixed effective positions within a program (e.g., the PIPELINE pragma, applicable to any loop). Moreover, many pragmas have a wide range of parameter values, and even those without

979-8-3315-7769-8/26/$31.00 ©2026 IEEE

explicit parameters implicitly include on/off settings indicating their activation status. These factors lead to an extremely large design space even for simple programs. Furthermore, the size of this space is sensitive to variations in program logic complexity and the types of pragmas employed: minor changes to either can significantly impact its scale. This raises the following challenges:

- **How to define the design space:** Including all pragmas results in a large design space, necessitating a trade-off between exploration speed and design quality. In fact, a small subset of pragmas drives most performance improvements (consistent with the Pareto principle), requiring careful selection of pragmas to include.
- **How to prune the design space:** Even after excluding pragmas with minimal impact, the remaining design space remains excessively large. Some pragmas are mutually conflicting, and their combined application generates redundant or even invalid designs. Additionally, although HLS tools allow users to specify parameters for certain pragmas, arbitrary configurations may lead to similar issues. Thus, constraints on pragma usage are needed to further shrink the design space.
- **How to explore the design space:** Once defined, the design space requires efficient exploration algorithms to guide the search for high-quality designs, while rapid evaluation of candidate designs is also indispensable.

B. Traditional Design Space Definition

Loops are often the most time-consuming components of a program, so applying loop-optimization pragmas can significantly improve performance. Existing HLS DSE frameworks typically incorporate three classic pragmas (PIPELINE, UNROLL, and ARRAY_PARTITION) into their design spaces. Some frameworks further integrate additional pragmas, sacrificing exploration speed for superior results. A representative example is HGBO [9], which includes 9 pragmas in its design space.

C. Performance Pragma

The PERFORMANCE pragma is the latest loop-optimization pragma released by Vitis HLS [15]. It allows users to specify a $target\_ti$ to constrain the number of clock cycles between successive loop initiations and automatically infers the low-level pragmas (including PIPELINE, UNROLL, ARRAY_PARTITION, FLATTEN, and INLINE) to satisfy this constraint. The $target\_ti$ is defined as:

$$target\_ti = trip\_count \times II, \tag{1}$$

where $trip\_count$ denotes the number of loop iterations and II represents the loop initiation interval.

D. Problem Formulation

HLS DSE is a multi-objective optimization problem that requires balancing multiple metrics, typically performance, power, and area (PPA). Different designs for the same program may exhibit a dominance relation (**Definition 1**) in their PPAs. The goal of DSE is to obtain Pareto-optimal solutions (**Definition 2**), which collectively form the Pareto set, also known as the Pareto front.

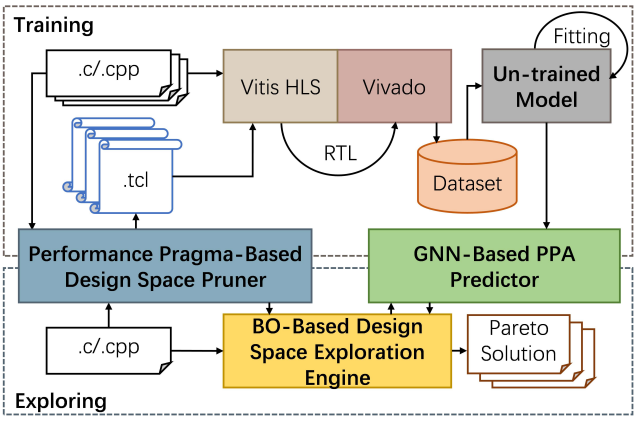

Fig. 1 The workflow of the proposed HLS DSE framework.

Definition 1 (Dominance relation)**.** *Let* $f : X \to \mathbb{R}^m$, $x \in X$, $f(x) = (f_1(x), f_2(x), \ldots, f_m(x))$. *Define* $f(x') \prec f(x^*)$ *if* $\forall i \in [1, m] : f_i(x') \leq f_i(x^*)$ *and* $\exists j \in [1, m] : f_j(x') < f_j(x^*)$.

Definition 2 (Pareto-optimal)**.** $x^* \in X$ *is Pareto-optimal if* $\nexists x' \in X : f(x') \prec f(x^*)$.

In the HLS DSE problem, each x represents a pragma combination, and $f_i(x)$ denotes a metric of the RTL design generated by that combination. Our goal is to obtain a high-quality set of Pareto-optimal solutions within an acceptable time by rationally exploring pragma combinations through design space pruning and efficient exploration algorithms, rather than exhaustively traversing the entire design space.

III. METHODOLOGY

A. Overview

Fig. 1 illustrates the proposed HLS DSE framework, which comprises three components: the design space pruner, the PPA predictor, and the design space exploration engine. The program to be explored is first input to the design space pruner, which analyzes the source code to identify potential placement locations and parameter values. Subsequently, the design space exploration engine explores the design space. It utilizes the PPA predictor to evaluate the quality of exploration points and thereby determine the next exploration direction. Exploration terminates when a specified time limit or iteration count is reached.

B. Design Space Pruner

In contrast to traditional design space definitions, we employ the PERFORMANCE pragma to replace classic loop-optimization pragmas. Additionally, we deduced the algorithmic logic underlying the PERFORMANCE pragma for allocating low-level pragmas through experiments. To the best of our knowledge, this is the first work to uncover and exploit the

deterministic mapping between a user-specified $target\_ti$ and the resulting set of low-level pragmas, thereby eliminating the exploration redundancy inherent in naive parameter sweeping. The PERFORMANCE pragma first analyzes the source code to extract loop characteristics (primarily $trip\_count$ and II), then sequentially determines whether to allocate pragmas to the current loop based on the user-specified $target\_ti$ and configures their parameters. The allocation algorithm is described in Alg. 1.

For an n-layer nested loop, loops are labeled from outer to inner as lp_1, lp_2, ..., lp_i, ..., lp_{n-1}, lp_n. The algorithm is simplified as follows: First, divide $target\_ti$ by lp_1's $trip\_count$. If the quotient is greater than or equal to lp_1's II, continue dividing the quotient by lp_2's $trip\_count$ and repeat the check until the quotient is less than lp_i's II. Then, completely unroll all inner loops of lp_i (lp_{i+1}, lp_{i+2}, ..., lp_n), and apply PIPELINE and UNROLL to lp_i such that:

$$target\_ti = \left\lceil \frac{trip\_count_i \times II_i}{\prod_{j=i}^{n} UNROLL\_factor_j} \right\rceil . \tag{2}$$

Equation (2) extends Equation (1), where $trip\_count_i$ denotes the iteration count of loop lp_i, II_i represents the initiation interval of loop lp_i, and $UNROLL\_factor_i$ is the parameter for the UNROLL pragma of loop lp_i.

Based on the above analysis, $UNROLL\_factor_i$ takes integer values within [1, $trip\_count_i$], and $target\_ti$ is also an integer. This implies that different user-specified $target\_ti$ values may cause the tool to infer the same $UNROLL\_factor_i$ (as verified in our experiments). Specifically, a single design result typically corresponds to a range of $target\_ti$ values, leading to redundant designs. Thus, the valid $target\_ti$ values (with one valid value assigned per redundant design group) form a set of disjoint integers. To determine these valid values, we traverse the $UNROLL\_factor_i$ values, starting from the PIPELINE stage of the innermost loop to the complete UNROLL stage of the outermost loop. Notably, large $UNROLL\_factor_i$ values may result in different $UNROLL\_factor_i$ corresponding to the same $target\_ti$ and the design space pruner automatically identifies such redundancy.

Algorithm 1 Low-Level Pragmas Allocation Algorithm

Input: A n-layer nested loop
Output: The n-layer nested loop with pragmas

1) Take the innermost loop lp_n
2) Analyze $trip\_count_n$ and II_n
3) Try in sequence:
   - Apply PIPELINE to lp_n
   - Apply PIPELINE and UNROLL to lp_n
   - Apply complete UNROLL to lp_n
4) **if** lp_n is a perfect nested loop **then**
   - Apply FLATTEN to lp_n
5) **if** fail to meet $target\_ti$ **then**
   - $n = n - 1$
   - **goto** 1)
6) Apply ARRAY_PARTITION
7) Apply INLINE
8) **return** lp_n

C. PPA Predictor

Beyond the vast design space, another key factor contributing to excessive exploration time is the time-consuming process of acquiring the actual QoR for RTL designs. In recent years, numerous academic studies have demonstrated that deep learning models for predicting the QoR of HLS-generated RTL designs are both feasible and effective [16].

As illustrated in Fig. 1, the PPA predictor employs GNN for program feature extraction. GNN aggregates features from neighbor nodes, enabling them to capture node dependencies. Thus, by extending the control data flow graph (CDFG) generated from C/C++ codes, we can predict the quality of designs produced by different pragma combinations [7].

Initially, C/C++ code is converted into LLVM IR [17] via Clang, which serves as the input for ProGraML [18]. As a widely adopted program representation specifically designed for machine learning, ProGraML is essentially an extended variant of the CDFG. We then augment this representation with pragma nodes and auxiliary nodes to facilitate the capture of program semantics, followed by the addition of necessary edges based on the inherent semantic meanings of each node type.

The PPA predictor utilizes TransformerConv [19] as its GNN layer. Compared to traditional GNN layers (e.g., GCN [20], GAT [21]), the self-attention mechanism of TransformerConv allows it to capture more complex inter-node dependencies by dynamically adjusting attention coefficients. The node update process is detailed as follows:

$$x_i' = W_1 x_i + \sum_{j \in N(i)} \alpha_{i,j} (W_2 x_j + W_6 e_{ij}), \tag{3}$$

where x_i denotes the embedding of node i, $e_{i,j}$ represents the edge embedding between nodes i and j, W_1, W_2, and W_6 are learnable weight matrices, and $\alpha_{i,j}$ is the attention coefficient between nodes i and j, defined as follows:

$$\alpha_{i,j} = \text{softmax}\left(\frac{(W_3 x_i)^T (W_4 x_j + W_6 e_{ij})}{\sqrt{D}} \right), \tag{4}$$

where W_3, W_4, and W_6 are learnable weight matrices, and D denotes the dimension of the node embedding.

To capture distant dependencies between nodes while avoiding over-smoothing, we employ the jumping knowledge network [22] to aggregate outputs from GNN layers at different depths. Global attention pooling [23] is then applied to derive the program's graph embedding, which is ultimately fed into a multi-layer perceptron (MLP) for predicting the actual PPA.

D. Design Space Exploration Engine

Bayesian optimization (BO) is a probabilistic framework for multi-objective optimization. It constructs a surrogate model (typically a Gaussian process) to probabilistically characterize the objective function. Its core resides in iteratively selecting exploration points through an acquisition

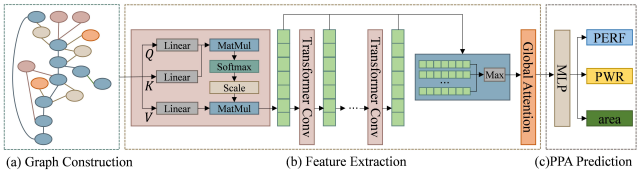

Fig. 2 The architecture of our PPA predictor.

function (AF), which balances exploitation (targeting high-quality candidate regions) and exploration (targeting under-explored regions) [24]. We adopt the expected improvement (EI) as the AF, defined as follows:

$$EI(x) = \int_{-\infty}^{+\infty} \max(y^* - y, 0) p(y \mid x)\, dy, \tag{5}$$

where y^* denotes the best observed objective value among the currently evaluated points, and y represents the objective value at the candidate point x, with $p(y \mid x)$ being the predictive distribution from the GP.

IV. EXPERIMENT

A. Experimental Setup

10 kernels were selected from the MachSuite benchmark for the experiments. For training the PPA predictor, 5 additional kernels were chosen from PolyBench, with 500 random pragma combinations generated per kernel. The HLS flow was executed using Vitis HLS 2025.1, and actual PPA were obtained via Vivado 2025.1, targeting Xilinx Virtex-7 VC707 part xc7vx485tffg1761-2 at 100 MHz. The automatic PIPELINE, UNROLL, and ARRAY_PARTITION were disabled during the flow. The PPA predictor employs 6 TransformerConv layers and was trained for 500 epochs with a batch size of 64, using the Adam optimizer with an initial learning rate of 0.001.

All experiments were conducted on Ubuntu 24.04.1 LTS, equipped with an Intel(R) Xeon(R) Platinum 8383C CPU and 4 NVIDIA GeForce RTX 3090 GPUs.

B. Experimental Results

Our approach is compared against HLS DSE methods based on traditional design space definitions, including simulated annealing (SA) [25], the non-dominated sorting genetic algorithm (NSGA-II) [26], and HGBO (the current state-of-the-art in HLS DSE) [9]. Specifically, we adopt HGBO's MOTPE-FL variant, which yields the best reported results. For fair comparison, the exploration time allocated to each benchmark matches that used in HGBO's experimental setup, with runtime measured in minutes. Design quality is evaluated using PPA metrics extracted from Vivado implementation reports: post-implementation clock period (CP) for performance, total on-chip power (in watts) for power, and area computed as the sum of normalized utilization rates of LUTs, FFs, DSPs, and BRAMs. The quality of the resulting Pareto front is quantified using the average distance from the reference set (ADRS) [27], defined as:

$$\text{ADRS}(\Gamma, \Omega) = \frac{1}{|\Gamma|} \sum_{\gamma \in \Gamma} \min_{\omega \in \Omega} f(\gamma, \omega), \tag{6}$$

TABLE I ADRS Comparison of the Pareto Fronts

Benchmark	Runtime(minutes)	SA	NSGA-II	HGBO	Ours
aes	31	**0.0753**	0.6531	0.1546	0.1272
bfs_bulk	17	0.5173	0.6385	0.3768	**0.1334**
fft_strided	18	0.4931	0.4999	0.3011	**0.2630**
gemm_ncubed	22	0.7612	0.7079	0.5517	**0.0529**
md_knn	29	0.6455	0.4073	0.2306	**0.1986**
nw	32	0.3627	0.4103	0.1457	**0.1106**
sort_radix	37	0.0001	0.0173	**0.0001**	0.0001
spmv_ellpack	20	0.6309	0.2895	0.1249	**0.0999**
stencil3d	33	0.5348	0.5659	0.3495	**0.3192**
viterbi	70	0.6643	0.6535	0.3366	**0.2821**
Average	31	0.4685	0.4843	0.2572	**0.1587**
Improv. over SA					**66.13%**
Improv. over NSGA-II					**67.23%**
Improv. over HGBO					**38.30%**

where the reference set consists of Pareto-optimal solutions aggregated from the exploration results of all compared methods.

As shown in TABLE. I, our framework achieves a 66.13% improvement in ADRS over SA, 67.23% improvement over NSGA-II, and 38.30% improvement over HGBO. These results demonstrate that, compared to HLS DSE frameworks based on traditionally defined design spaces, our approach significantly enhances both exploration quality and efficiency through the proposed design space pruning strategy and the GNN-based PPA predictor.

V. CONCLUSION

This work replaces traditional loop-optimization pragma combinations with the PERFORMANCE pragma. By deducing the algorithmic logic underlying the PERFORMANCE pragma, we further prune the design space, achieving a significant reduction in its scale. To validate the exploration efficiency and solution quality of the pruned design space, we propose an HLS DSE framework integrating an accurate GNN-based PPA predictor and an efficient BO-based design space exploration engine. Experimental results demonstrate that our framework obtains superior Pareto-optimal sets compared to state-of-the-art methods under the same time constraint.

Future research will expand the current design space by incorporating pragmas of different optimization granularities, enabling comprehensive program optimization and more flexible trade-offs between exploration speed and solution quality. Additionally, we will further explore better QoR prediction models and DSE methodologies to enhance the performance of our framework.

ACKNOWLEDGMENT

This work is supported by the National Natural Science Foundation of China (No. 62404021, 62404257), 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), Shenzhen Science and Technology Program (No. JCYJ20250604175310014).

REFERENCES

[1] I. Kuon, R. Tessier, and J. Rose, *FPGA Architecture: Survey and Challenges*, 2008.

[2] Y. Hu, Y. Du, E. Ustun, and Z. Zhang, "GraphLily: Accelerating Graph Linear Algebra on HBM-Equipped FPGAs," in *2021 IEEE/ACM International Conference On Computer Aided Design (ICCAD)*, 2021, pp. 1–9.

[3] J. Cong, J. Lau, G. Liu, S. Neuendorffer, P. Pan, K. Vissers, and Z. Zhang, "FPGA HLS Today: Successes, Challenges, and Opportunities," *ACM Trans. Reconfigurable Technol. Syst.*, vol. 15, no. 4, 2022.

[4] R. Hou, J. Zhai, Y. Wang, Z. Lin, and K. Zhao, "Array Partitioning Method for Streaming Dataflow Optimization in High-level Synthesis," in *2024 2nd International Symposium of Electronics Design Automation (ISEDA)*, 2024, pp. 278–282.

[5] 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*, vol. 39, no. 10, pp. 2628–2639, 2020.

[6] A. Sohrabizadeh, C. H. Yu, M. Gao, and J. Cong, "AutoDSE: Enabling Software Programmers to Design Efficient FPGA Accelerators," *ACM Trans. Des. Autom. Electron. Syst.*, vol. 27, no. 4, 2022.

[7] A. Sohrabizadeh, Y. Bai, Y. Sun, and J. Cong, "Robust GNN-Based Representation Learning for HLS," in *2023 IEEE/ACM International Conference on Computer Aided Design (ICCAD)*, 2023, pp. 1–9.

[8] 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*, vol. 42, no. 3, pp. 900–913, 2023.

[9] 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.

[10] Z. Zou, C. Tang, L. Gong, C. Wang, and X. Zhou, "FlexWalker: An Efficient Multi-Objective Design Space Exploration Framework for HLS Design," in *2024 34th International Conference on Field-Programmable Logic and Applications (FPL)*, 2024, pp. 126–132.

[11] Y. Bai, A. Sohrabizadeh, Z. Ding, R. Liang, W. Li, D. Wang, H. Ren, Y. Sun, and J. Cong, "Learning to Compare Hardware Designs for High-Level Synthesis," in *Proceedings of the 2024 ACM/IEEE International Symposium on Machine Learning for CAD*, 2024.

[12] Z. Qin, Y. Bai, A. Sohrabizadeh, Z. Ding, Z. Hu, Y. Sun, and J. Cong, "Cross-Modality Program Representation Learning for Electronic Design Automation with High-Level Synthesis," *Proceedings of the 2024 ACM/IEEE International Symposium on Machine Learning for CAD*, 2024.

[13] Z. Wu, S. Pan, F. Chen, G. Long, C. Zhang, and P. S. Yu, "A Comprehensive Survey on Graph Neural Networks," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 32, no. 1, pp. 4–24, 2021.

[14] N. Wu, H. Yang, Y. Xie, P. Li, and C. Hao, "High-level synthesis performance prediction using GNNs: benchmarking, modeling, and advancing," in *Proceedings of the 59th ACM/IEEE Design Automation Conference*, 2022.

[15] AMD, "UG1399," https://docs.amd.com/r/en-US/ug1399-vitis-hls.

[16] H. Ren, S. Nath, Y. Zhang, H. Chen, and M. Liu, "Why are Graph Neural Networks Effective for EDA Problems?" in *2022 IEEE/ACM International Conference On Computer Aided Design (ICCAD)*, 2022, pp. 1–8.

[17] C. Lattner and V. Adve, "LLVM: a compilation framework for lifelong program analysis transformation," in *International Symposium on Code Generation and Optimization, 2004. CGO 2004.*, 2004, pp. 75–86.

[18] C. Cummins, Z. V. Fisches, T. Ben-Nun, T. Hoefler, and H. Leather, "ProGraML: Graph-based Deep Learning for Program Optimization and Analysis," 2020.

[19] Y. Shi, Z. Huang, S. Feng, H. Zhong, W. Wang, and Y. Sun, "Masked Label Prediction: Unified Message Passing Model for Semi-Supervised Classification," 2021.

[20] T. N. Kipf and M. Welling, "Semi-Supervised Classification with Graph Convolutional Networks," 2017.

[21] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Liò, and Y. Bengio, "Graph Attention Networks," 2018.

[22] K. Xu, C. Li, Y. Tian, T. Sonobe, K. ichi Kawarabayashi, and S. Jegelka, "Representation Learning on Graphs with Jumping Knowledge Networks," 2018.

[23] Y. Li, D. Tarlow, M. Brockschmidt, and R. Zemel, "Gated Graph Sequence Neural Networks," 2017.

[24] C. Bai, Q. Sun, J. Zhai, Y. Ma, B. Yu, and M. D. Wong, "BOOM-Explorer: RISC-V BOOM Microarchitecture Design Space Exploration Framework," in *2021 IEEE/ACM International Conference On Computer Aided Design (ICCAD)*, 2021, pp. 1–9.

[25] A. Mahapatra and B. C. Schafer, "Machine-learning based simulated annealer method for high level synthesis design space exploration," *Proceedings of the 2014 Electronic System Level Synthesis Conference (ESLsyn)*, pp. 1–6, 2014.

[26] B. C. Schafer, "Parallel High-Level Synthesis Design Space Exploration for Behavioral IPs of Exact Latencies," *ACM Trans. Des. Autom. Electron. Syst.*, vol. 22, no. 4, 2017.

[27] 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*, vol. 39, no. 10, pp. 2628–2639, 2020.