Received 6 November 2024, accepted 24 November 2024, date of publication 2 December 2024, date of current version 23 December 2024.

Digital Object Identifier 10.1109/ACCESS.2024.3509606

RESEARCH ARTICLE

Hybrid Graph Representation and Learning Framework for High-Level Synthesis Design Space Exploration

POUYA TAGHIPOUR [1], **(Student Member, IEEE), ERIC GRANGER** [2], **(Member, IEEE), AND YVES BLAQUIÈRE** [1], **(Member, IEEE)**

[1]LaCIME, Department of Electrical Engineering, École de Technologie Supérieure (ÉTS), Montreal, QC H3C 1K3, Canada
[2]LIVIA, ILLS, Department of Systems Engineering, École de Technologie Supérieure (ÉTS), Montreal, QC H3C 1K3, Canada

Corresponding author: Pouya Taghipour (pouya.taghipour.1@ens.etsmtl.ca)

This work was supported by the Natural Sciences and Engineering Research Council (NSERC) of Canada.

ABSTRACT Optimizing hardware accelerators in high-level synthesis (HLS) relies on design space exploration (DSE), which involves experimenting with different pragma options and trading off hardware cost and performance metrics (HCPMs) to identify Pareto-optimal solutions. The exponential growth of the design space, poor quality-of-results (QoR) estimation by HLS tools, and lengthy post-implementation runtime have made the HLS DSE process highly challenging and time-consuming. Automating this process could reduce time-to-market and associated development costs. Learning-based methods, particularly graph neural networks (GNNs), have shown considerable potential in addressing HLS QoR/DSE problems by modeling the mapping function from control data flow graphs (CDFGs) of HLS designs to their logic, enabling early estimation of QoR during the compilation phase of the hardware design flow. However, there is still a gap in terms of their prediction accuracy. Indeed, modeling HLS-related problems using GNNs that efficiently capture the complex patterns arising from applied pragmas and low-level characteristics of HLS specifications is challenging. This paper introduces a novel hybrid graph representation and learning framework under a multi-task setting, featuring two distinct types of CDFGs derived from two different sources. Furthermore, various models are proposed to fuse features and knowledge in joint, sequential, and parallel learning architectures, aiming to improve the overall accuracy and generalization in predicting QoR and approximating the Pareto frontier (PF). Experimental results show that our framework can attain a higher level of performance than the state-of-the-art baseline models over unseen designs, with an average relative improvement of 47.4 % and 16.0 % for resource utilization and performance metrics, respectively. Additionally, considering various HLS designs with different design space sizes, a 26.8 % enhancement in DSE PF approximation is observed.

INDEX TERMS Electronic design automation (EDA), high-level synthesis (HLS), design space exploration (DSE), machine learning (ML), graph neural networks (GNN), field-programmable gate array (FPGA).

I. INTRODUCTION

Hardware acceleration [1], [2] based on *application-specific integrated circuits (ASICs)* and *field-programmable gate arrays (FPGAs)* has gained particular importance in recent years due to the obsolescence of Moore's law [3] and the breakdown of Dennard Scaling [4] era. This approach has emerged to meet the rising demand for high-performance and energy-efficient computing platforms driven by the growth of compute-intensive and data-driven applications across various industries. The primary issue with this technology, though, is the long design time and correspondingly high costs. One potential solution involves enhancing existing *electronic design automation (EDA)* tools at various stages of the hardware design flow (e.g., behavioral description, logic synthesis, verification, etc.) with the ultimate goal of reducing

The associate editor coordinating the review of this manuscript and approving it for publication was Harikrishnan Ramiah.

the development cycle from months and years to the shortest feasible duration [5].

Hardware description languages (HDLs) like VHDL [6] and Verilog [7] have been the mainstream approach for hardware accelerator design at the *register-transfer level (RTL)*. However, using HDLs for modern algorithms with increased complexity and scale is challenging and labor-intensive.

High-level synthesis (HLS) [8], [9], [10], [11], [12] has significantly improved the development process of hardware accelerators by raising the abstraction level from the RTL to the behavioral description. Indeed, HLS allows hardware functionalities to be described in untimed or loosely-timed high-level languages (e.g., C/C++/SystemC), which are more understandable, verifiable, and portable. Besides boosting design productivity, a key aim of HLS has been to break the monopoly traditionally held by hardware designers in developing domain-specific accelerators.

Also, HLS enables users to create a multitude of configurations for a design, leveraging diverse *optimization directives*, aka *pragmas*. Designers can specify pragmas for various code components e.g., loop pipelining/unrolling, memory partitioning/merging, and function inlining/pipelining [13], [14] without the need to change the high-level description code. Consequently, the HLS tool is guided to manage parallelization, memories/resource types, and other factors w.r.t the design specifications and product type. Corresponding to configurations, micro-architectures are generated, maintaining identical functionality but differ greatly in terms of hardware costs and performance metrics (HCPMs). The costs usually encompass metrics in the form of on-chip resource utilization, including the number of look-up tables (LUTs), flip-flops (FFs), digital signal processor units (DSPs), block RAMs (BRAMs), and configuration logic blocks (CLBs). Also, the most important performance measures, include critical path timing (CP), which determines the maximum frequency of the generated datapath, latency as the total number of clock cycles (CCs) to perform one complete task, and execution time, calculated as $T_{\text{exe}} = \text{CP} \times \text{CC}$.

Despite the considerable advantages of HLS, fundamental challenges impede its wider adoption in the industry. More specifically:

(1) The hardware design flow is inherently complex and time-consuming. While the compilation, building, and execution of a software function targeting a processor can be completed in a few seconds, a similar process for a single HLS design entry targeting FPGAs can take minutes to hours. This is due to the need for designs to go through the HLS, post-synthesis (logic synthesis), and post-implementation (includes translation, technology mapping, and place&route) phases.

(2) The intermediate estimation of *quality-of-results (QoR)* provided by current HLS tools is not sufficiently reliable and accurate [15]. Nevertheless, the HLS stage itself takes a few to dozens of minutes to complete concerning design size, complexity, and applied pragmas, making agile HCPM trade-offs challenging.

(3) Achieving highly efficient designs relies on *design space exploration (DSE)* [16], [17], which entails iteratively re-tuning pragmas, and trading off HCPMs. The main challenge is that the size of the design space expands exponentially with the inclusion of more pragmas and the range of exploration values assigned to each. Therefore, following the brute forcing approach makes exhaustive exploration impractical due to the non-negligible run-time of the post-implementation processes, which can take up to hours and days even for small to medium-sized designs [18], [19]. In addition, DSE is a *multi-objective optimization problem (MOOP)*, where only a few solutions are considered optimal in terms of HCPMs. Taking into account that conflicts exist among different design objectives, For example, reducing latency, while minimizing area.

(4) The impact of pragmas on the final datapath is highly complex and depends on many factors [19]. Determining the appropriate pragma, or combination thereof, along with their values with current HLS tools, necessitates designer intervention. This demands highly specialized knowledge and extensive experience.

Over the past decade, a considerable amount of research has been devoted to automating the HLS DSE process. A predominant portion of prior research works falls into categories *analytical model-based* and *AI-assisted* methods. Conventionally, analytical model-based DSE methods [20], [21], [22], [23], [24], [25], [26], [27] rely on human effort to model and formulate the knowledge that describes the relationship between inputs (code and pragmas) and HCPMs. On the other hand, AI-assisted methods are capable of acquiring the knowledge automatically without human intervention during (online) or before (offline) exploration of design space [28], based on *refinement-based* [29], [30], [31], [32], [33] and *machine learning (ML)-based* [18], [34], [35], [36] methods, respectively.

Analytical model-based DSE methods are fast since they don't require HLS invocation, and leverage the knowledge initially embedded in the prediction model. However, such techniques have been provided for regular code structures [20], [21], [22] with supporting a limited number of pragmas [20], [24] or developed for only a particular type of applications [26], [27]. Lin-Analyzer [23], FlexCL [24], COMBA [25], and MPSeeker [37] presented more comprehensive frameworks. However, these methods exhibit generality, model management, and maintenance limitations [38], [39], [40]. For example, in the case of updating an HLS tool, the knowledge embedded into the model needs to be re-defined. Furthermore, HLS tools show highly complex behavior when multiple pragmas are applied simultaneously, due to the inter-dependencies of pragmas and the interactions between low-level data and operations in HLS specifications. Modeling such scenarios analytically for a diverse range of real-world applications is questionable.

Refinement-based DSE methods acquire knowledge by continuously learning from design space through sampling, processing the feedback, and intelligently guiding the algorithm to the optimal solutions [28]. Such models are refined iteratively by focusing more on a sub-region of design space to find Pareto-optimal cases. The primary limitation of these methods lies in the need for several invocations of the HLS tool during the DSE process. Furthermore, as the refinement model runs independently for each design, re-using and transferring the learned knowledge are problematic.

ML algorithms enable computers to learn complex mapping functions from data by identifying patterns and correlations within them. In ML-based DSE methods, the knowledge is gained before starting the DSE process in a data-driven training process. These models can subsequently be employed to predict HLS QoR. As an important advantage, they can acquire knowledge automatically, unlike analytical model-based methods, while providing the same capability in terms of prediction speed. Additionally, the acquired knowledge is transferable to another design and/or hardware technology as demonstrated in [36], [41], and [42]. In the domain of ML-based DSE methods, graph learning [40], [43], [44] has shown substantially better prediction performance compared to conventional ML models. This has been achieved by representing programs as graphs, particularly, *control data flow graphs (CDFGs)* [8], [45] and learning CDFG-to-logic mapping function using *graph neural networks (GNNs)* [46], [47]. GNNs are powerful tools that can capture both semantics and structural information within graphs and encode the information to the lower dimension, facilitating downstream prediction tasks.

There are two main forms of CDFGs used to represent HLS specifications. The first form is constructed using generic software compilers targeting CPUs, while the second form can be extracted from the front end of HLS compilers targeting specific hardware. Among state-of-the-art (SOA) works, [43] proposes a framework based on compiling the HLS code with a generic software compiler, generating CDFGs from the intermediate representation (IR), and training a GNN model to learn over them. However, this work only investigated the loop unrolling pragma. In a similar approach, gnn4hls [40] was introduced, supporting a broad spectrum of optimization directives.

It is worth noting that generic software compilers cannot process HLS pragmas. As a result, CDFGs with identical structure and size are used for all configurations of a given HLS design. To tailor these CDFGs for HLS problems, the incorporation of pragma information as a dedicated pragma node within the CDFGs was proposed in [43]. In contrast, the gnn4hls [40] integrates pragma information directly into the node features of the CDFGs. Also, in [15], pragma information was used as global graph attributes. Whilst, in typical HLS tools, HLS specifications undergo various hardware-specific optimizations and transformations [45]. Consequently, different configurations of an HLS design result in CDFGs that differ in size, structure, and features. For example, applying a loop unrolling pragma results in CDFGs that contain copies of the loop body w.r.t the unrolling factor. The Program-to-Circuit [44] utilized this form of CDFGs, but it did so only for random and real-case functions without considering any HLS pragmas.

Subsequent studies have shown that relying solely on CDFGs is insufficient to achieve a high-accuracy model using GNNs. For example, auxiliary nodes were introduced in [48] to integrate program semantics and pragmas by providing high-level hierarchical information. In [49], HLS intermediate results were used as supplement information.

Despite significant advancements in leveraging GNNs for HLS QoR/DSE challenges, a noticeable gap still exists in achieving the level of prediction accuracy required by the EDA industry [50]. This shortfall highlights the need for further research and innovation to bridge this gap and enhance the accuracy and generalization of GNN-based DSE automation tools. In contrast with previous works, this research addresses two primary emerging questions. *Q1: Which form of CDFGs are more suitable for GNN-based models in the context of predicting different HCPMs in HLS? Q2: Can a more accurate model be achieved by integrating CDFGs from two distinct sources and/or fuse the knowledge in hybrid models?*. To do that, a novel hybrid graph representation and learning framework is introduced by emphasizing both HLS data representation and prediction model to enhance DSE results using GNNs. The key contributions of this paper are as follows:

A. STUDYING THE IMPACT OF HLS CDFG TYPES

Two distinct forms of CDFGs are explored to represent HLS designs, derived from software-based and hardware-based compilers. Our investigation reveals each yields different prediction accuracy for different HCPMs when applied with GNNs. This assessment is conducted during training as well as in transfer learning on entirely unseen HLS designs.

B. NOVEL HYBRID GRAPH LEARNING MODELS

Leveraging multi-task architecture, three different hybrid GNN models are proposed including 1)*jointly-learning and fusion*, 2)*sequential-learning with knowledge infused*, and 3)*parallel learning and fusion* models. Experimental results show that our hybrid GNN models not only surpass the baselines (single GNN model with a single type of CDFGs as input) and the HLS tool (*VitisHLS*) in terms of prediction accuracy but also exhibit the higher capability of generalization to previously unseen designs (out of our training dataset) by extending the learned knowledge.

C. ENHANCED DSE APPROXIMATION

Integrating the proposed hybrid GNN models for HLS QoR prediction into our HLS DSE automation framework shows enhanced capabilities in DSE Pareto frontier (PF) approximations compared to baseline models and commercial HLS tools. This improvement is shown by evaluations of designs from various domains (image processing, searching, sorting algorithms, etc.) and with diverse configuration sizes.

The rest of the paper is organized as follows. Section II presents a theoretical problem formulation of the HLS DSE. Section III outlines the proposed methodology, detailing the two different forms of CDFGs, proposed multi-task, and hybrid graph learning models. Section IV provides the experiments setup, dataset, evaluation procedure, and obtained results for HLS QoR estimation and DSE.

II. PROBLEM FORMULATION

This section presents the theoretical formulation of the HLS DSE in the context of the discrete MOOP.

Let S be an HLS specification, which principally is a synthesizable format of a design developed in a high-level programming language, aiming at being implemented on hardware. Considering H as an HLS vendor tool followed by the post-synthesis and post-implementation processes, S is directly received by H as the primary input. Additionally, some global knobs (aka global constraints) are determined, such as the target clock period and hardware technology, represented by K_{CP} and K_{HW}, respectively. These constraints impact the entire behavioral description [16], having a fixed value in this research across all designs. Moreover, a set of local knobs is defined in an encoded form of $\langle$ code component/label name, pragma type (e.g., loop unrolling), and pragma setting (e.g., factor $= 2$) $\rangle$, denoted by K_{P}. Code. 1 shows *stencil2d* HLS specification from MachSuit benchmark [51] and a set of example local knobs associated with this function is given in Table 1.

Let $\boldsymbol{d} = (d_1, d_2, \cdots, d_n) \in \mathcal{D}$ be a design vector, where d_i is a design variable that specifies the value for the i-th knob ($k_i \in K_P$), n is the total number of design variables (which is equal to the number of local knobs), the $\mathcal{D}$ is the design space (aka solution space). Accordingly, $N = |\mathcal{D}|$ represents the size of design space, which is equivalent to the Cartesian product among all setting sets within K_P.

Given $\boldsymbol{d}, K_{\mathrm{CP}}$ and K_{HW}, synthesizing S by H forms a mapping described as $H : (S, \boldsymbol{d}, K_{\mathrm{CP}}, K_{\mathrm{HW}}) \rightarrow \boldsymbol{y}$, such that $\boldsymbol{y} = (y_1, y_2, \cdots, y_m) = (f_1(\boldsymbol{d}), f_2(\boldsymbol{d}), \cdots, f_m(\boldsymbol{d}))$ is the target function consisting of m objective functions that are to be optimized simultaneously. An example of this mapping formulation, having two design variables is illustrated in Fig. 1. Consequently, the exhaustive implementation of S over $\mathcal{D}$ is defined as $H : (S, \mathcal{D}, K_{\mathrm{CP}}, K_{\mathrm{HW}}) \rightarrow \mathcal{Y}$, where $\mathcal{Y} = \{\boldsymbol{y}_1, \boldsymbol{y}_2, \cdots, \boldsymbol{y}_N\}$ is objective space as a result of the projection of the design space. In reality, $|\mathcal{Y}| \leq |\mathcal{D}|$ because applying some $\boldsymbol{d} \in \mathcal{D}$ to S may not be applicable or lead to a successful implementation. Lastly, the optimization goal is defined as minimizing a target function $\boldsymbol{y}$ as (1). Although, this study focuses on a dual-objectives DSE task, in which different resource utilization include the number of LUTs, FFs, DSPs or BRAMs served as a cost function, denoted by f_c while CP, CC, and T_{Exe} serve as the performance function, denoted by f_{p}.

$$\underset{\mathcal{D}}{\operatorname{argmin}}\, \mathcal{Y} = \boldsymbol{f}(\boldsymbol{d}) = \Big(f_1(\boldsymbol{d}), f_2(\boldsymbol{d}), \cdots, f_m(\boldsymbol{d})\Big) \tag{1}$$

Among all $\boldsymbol{d} \in \mathcal{D}$, the designer is interested in a subset of $\mathcal{D}$ named solution set $\mathcal{D}^*$, containing Pareto design vectors (d^*) that result in Pareto-optimal implementations. The $\mathcal{D}^*$ is defined as:

$$\mathcal{D}^* = \{\boldsymbol{d} \mid \boldsymbol{d} \in \mathcal{D} \text{ and } \boldsymbol{d} \text{ is Pareto}\} \tag{2}$$

Formally, $\boldsymbol{d}^* \in \mathcal{D}$ is said to be Pareto optimal if and only if there does not exist any other $\boldsymbol{d} \in \mathcal{D}$ that dominate it, i.e.:

$$f_{\mathrm{c}}(\boldsymbol{d}) \leq f_{\mathrm{c}}(\boldsymbol{d}^*) \text{ with at least one strict inequality, and}$$
$$f_{\mathrm{p}}(\boldsymbol{d}) \leq f_{\mathrm{p}}(\boldsymbol{d}^*) \text{ with at least one strict inequality.}$$

CODE 1 Stencil2d HLS specification (S) from MachSuite [51] benchmark

```
void stencil2d (int orig[N*C], int sol[N*C], int filter[f_size])
{
  int temp, mul;
  L1:for(int r=0; r<N-2; r++) {
    L2:for(int c=0; c<C-2; c++) {
      temp = (int)0;
      L3:for(int k1=0; k1<3; k1++) {
        L4:for (int k2=0; k2<3; k2++) {
          mul = filter[k1*3 + k2]*orig[(r+k1)*C+ c+k2];
          temp += mul;
        }//end L4
      } //end L3
      sol[(r*C) + c] = temp;
    } //end L2
  } //end L1
}
```

TABLE 1. The list of local knobs (K_P) for Stencil2d HLS specification. The design space size for this example is 4096.

K_{P}	Component	Pragma Type	Pragma Setting

k_1, k_2	orig, sol	Array Partitioning	dim={1}
			type={cyclic, block}
			factor={4,16,32,64}

k_3	filter	Array Partitioning	type={complete}
k_4, k_5	L3, L4	Loop Unrolling	factor={1,2,3,4}
k_6, k_7	L3, L4	Loop Pipelining	factor={On, Off}
k_8	mul	Resource	core=DSP

The Pareto front (PF) is a set of non-dominated solutions in the objective space, showing the trade-offs between the conflicting objectives. Consequently, answers to such problems should aim to find $\mathcal{D}^*$ and approximate the PF rather than a single solution. An example of first-rank PF is depicted as dashed red lines in Fig. 1.

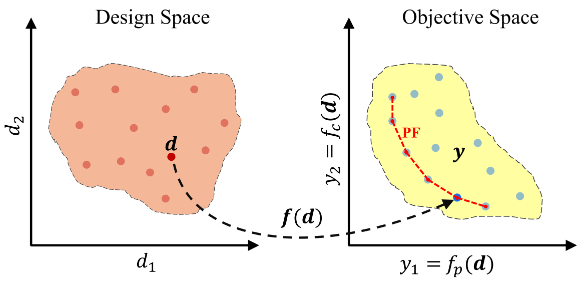


FIGURE 1. An example 2-dimensional mapping from design space for a design vector $\boldsymbol{d}$ based on d_1 and d_2 pragma values (served as two design variables) to the objective space, considering a performance (f_p) and a cost (f_c) functions. The true PF is shown by a dashed red line that needs to be approximated accurately in the HLS DSE process.

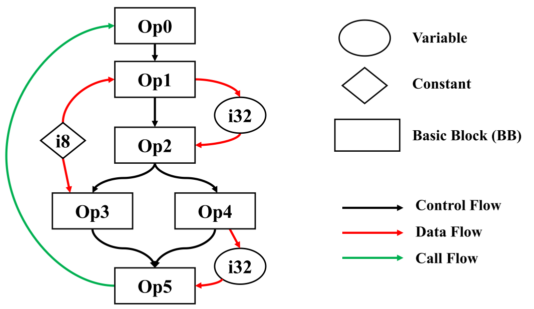


FIGURE 2. An abstract CDFG example with different types of nodes and edges is illustrated.

III. PROPOSED METHODOLOGY

Accurately approximating the PF in the HLS DSE process requires precise prediction of the QoR. Focusing on enhancing the accuracy of GNN-based HLS QoR/DSE prediction models, this section provides a detailed description of the proposed framework, including methods for representing HLS code as CDFGs, the baseline multi-task prediction model, and three different fusion-based GNN models.

A. METHODS FOR REPRESENTING HLS SPECIFICATIONS AS GRAPHS

Employing graphs as a language-independent structure has been a natural and efficient way to represent programs in the front end of compilers, simplifying analysis and optimization processes without directly involving syntax and rules of codes [45].

Mathematically, a graph is expressed as $G = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V} = \{\upsilon_1, \upsilon_2, \upsilon_3, \ldots, \upsilon_\nu\}$ is the set of vertices (or nodes), and $\mathcal{E}$ is the set of edges (or links). Let $\upsilon_i, \upsilon_j \in \mathcal{V}$, then $e_{ij} = (\upsilon_i, \upsilon_j) \in \mathcal{E}$ denotes an edge from υ_i to υ_j. The neighborhood of a node υ is defined as $\mathcal{N}(\upsilon) = \{u \in \mathcal{V} \mid (u, \upsilon) \in \mathcal{E}\}$, $\mathcal{N}(\upsilon) \subseteq \mathcal{V}$. For a finite graph with $\nu = |\mathcal{V}|$ nodes, there exists an $\nu \times \nu$ adjacency matrix ($\boldsymbol{A}$), also called a connection matrix. The elements of $\boldsymbol{A}$ obtain 1 or 0 values, depending on whether a direct path exists between pairs of vertices or not. Each node $\upsilon \in \mathcal{V}$ possesses an initial feature vector with a dimension of a, denoting as $\boldsymbol{x}_\upsilon \in \mathbb{R}^a$. Hence, $\boldsymbol{X} = \{\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_\nu\}$ is a set of feature vectors of nodes, stored as a matrix $\boldsymbol{X} \in \mathbb{R}^{\nu \times a}$ called graph node feature matrix. Conversely, $\boldsymbol{X}^e \in \mathbb{R}^{|\mathcal{E}| \times b}$ is graph edge feature matrix with $\boldsymbol{x}^e_{u,\upsilon} \in \mathbb{R}^b$ representing the feature vector of the edge (u, υ) with size b.

One standard abstract representation of codes is CDFGs, formed by combining control flow graphs (CFGs) and data flow graphs (DFGs). CDFGs are directed graphs in which nodes represent the *basic blocks (BB)* of a program, and the edges indicate both the flow of control between BBs (i.e., the program's execution sequence) and the dependencies between data elements. An abstract example of a CDFG is presented in Fig. 2. In this diagram, different types of nodes are shown: basic blocks (BBs) as rectangles, variables as circles, and constants as rhombuses. The graph also includes various edges to indicate control, data, and call flows, shown as black, red, and green arrows, respectively. Each basic block (BB) contains a distinct operation code (Opcode), reflecting the function's low-level operations.

This research explores two different methods for modeling HLS data, each producing distinct forms of CDFGs. The first form is generated from a generic software compiler (in our case, Clang [52]), inspired by SOA works. The second form is obtained directly from the front end of an HLS compiler. Using the first method, CDFGs are generated with structures and sizes that remain independent of the applied pragmas across different configurations of a given HLS design. In contrast, the second method results in CDFGs with varying structures and sizes. To customize the first form of CDFGs for HLS-related problems, the pragma information is incorporated into the node feature vectors. This is while in typical HLS design flow, the abstract representation of the high-level functionalities undergoes several hardware-specific optimizations and loop transformations [45]. For example, partitioning a BRAM port by a factor of N results in the generation of N ports (each with 1/N the length of the original array) in the generated CDFGs, with the associated read/write operations duplicated for each array. Our framework exploits *VitisHLS* [53], an open-source HLS frontend to extract the post-IRs of HLS specifications and represent them as CDFGs. The proposed framework leverages both forms of CDFG in a hybrid manner. Throughout this paper, the former CDFGs are denoted with G_{sw} and the latter with G_{hw} that reflect the rich hardware-specific structural information.

The initial feature of nodes for G_{sw} and G_{hw} is detailed in Table 2. As shown, both share some common features, such as node type, opcode category, and data width. The "opcode" corresponds to the low-level virtual machine (LLVM) [54], [55] operation code (e.g., load, allocate, add), while the "opcode group" in this table denotes their categorization (e.g., memory operation, binary operation, etc.). In addition, the pragma feature in G_{sw} contains status and values for all applied pragma for each graph component. This feature is a vector with a size of 10. Overall, G_{sw} and G_{hw} have initial node feature sizes of 15 and 4 respectively. No global attributes are considered in the proposed CDFGs for HLS representation.

TABLE 2. List of node features.

Graph Type	Feature Name	Values
G_{sw}	Node type	Instruction, Variable, Constant
	Block ID	Integer
	Function ID	Integer
	Data Width (bits)	8,16,32, etc.
	Opcode Group	Conversion, Memory, etc.
	Port	True, False
	Pragma	Loop Unrolling, Dataflow, etc
G_{hw}	Node type	Instruction, Variable, Constant
	Port	True, False
	Data Width (bits)	8,16,32, etc.
	Opcode group	Conversion, Memory, etc.

The edge features of each form of CDFGs are given in Table 3. Accordingly, G_{sw} and G_{hw} have an edge feature vector with sizes of 3 and 2, respectively.

TABLE 3. List of edge features.

Graph Type	Feature Name	Value
G_{sw}	Edge Type	Control, Data, Call
	AreSameFunction	True, False
	AreSameBlock	True, False
G_{hw}	Edge Type	Control, Data, Call
	Edge Back	True, False

B. HLS RESOURCE AND PERFORMANCE PREDICTION MODEL BASED ON GRAPH NEURAL NETWORKS

Traditional neural networks are primarily able to handle vectorized (e.g, sound) or grid-like data structures (e.g., images), whereas graph-structured data is inherently irregular and non-Euclidean, making it challenging for such networks to process [56]. This challenge has been addressed by the introduction of GNNs, which are end-to-end deep learning models capable of performing various prediction tasks on graph data using the same feed-forward back-propagation structure.

Our methodology for HLS QoR prediction employs supervised graph-level regression, where a labeled set of training points (CDFG with available post-implementation resources and performance results) is used to train a model and learn a mapping function from input data points (CDFG) to output variables. In this regard, each graph in the training dataset is an independent and identically distributed (i.i.d) data point.

In graph-level prediction using GNNs, the objective is to encode of the input graph into a low-dimensional Euclidean space while simultaneously capturing the relevant contextual information. Indeed, GNNs optimize this learning to attain the most effective representative model such that the quality of the results is heavily reliant on the encoding process. For this reason, devising a method that effectively incorporates both local (individual nodes and their immediate neighbors) and topological information (the structural properties and relationships between nodes) in entire graph data is an important step in GNN models. The process of graph embedding relies on node embedding and pooling stages. The node embedding itself works based on *message-passing (MP)* framework, which generalizes spatial convolution on structure data through message propagation and recursively updates the features (hidden state) of nodes by exchanging information with their neighbors (neighbor aggregation). At each iteration (or layer) denoted by ℓ, an MP layer operation on target node u is defined as:

$$\begin{aligned} \boldsymbol{m}_{\mathcal{N}(u)}^{(\ell)} &= \Lambda_{\Theta}^{(\ell)}\left(\{\boldsymbol{h}_{\upsilon}^{(\ell)}, \forall \upsilon \in \mathcal{N}(u)\}\right) \\ \boldsymbol{h}_{u}^{(\ell+1)} &= \Psi_{\Theta}^{(\ell)}\left(\boldsymbol{h}_{u}^{(\ell)}, \boldsymbol{m}_{\mathcal{N}(u)}^{(\ell)}\right) \end{aligned} \quad (3)$$

where $\boldsymbol{h}_u^{(\ell+1)} \in \mathbb{R}^c$ is the hidden state of the target node u in the next iteration, and $\boldsymbol{h}_\upsilon^{(\ell)} \in \mathbb{R}^c$ is the hidden state of neighborhood nodes υ at layer ℓ. In both terms, c is the hidden feature size that in our implementation is considered the same across all layers. In addition, $\Lambda_\Theta^{(\ell)}$ and $\Psi_\Theta^{(l)}$ are arbitrary differentiable functions with learnable parameters, called *AGGREGATE* and *UPDATE*, respectively. Fig. 3 presents a conceptual representation of the message-passing mechanism applied to node 1 within the input graph. As shown, Each node in the graph is initially assigned a 4-dimension feature vector. During the message-passing phase, the neighboring nodes' feature vectors are aggregated through a designated aggregation function, which consolidates the relevant information. Subsequently, the aggregated data, in conjunction with the current state of node 1, is processed by an update function that refines the node's representation. This process results in an updated 8-dimension feature vector for node 1. The same procedure is applied for all nodes within the graph to propagate information and enhance node-level feature representations.

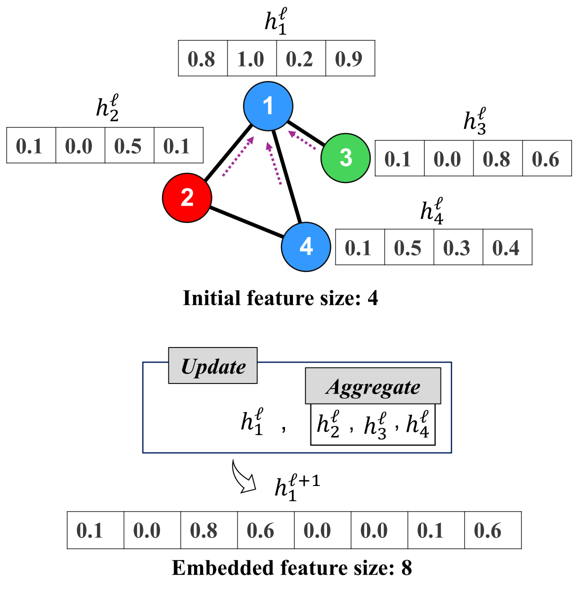


FIGURE 3. An abstract graphical example for message passing framework.

In graph-level prediction models, the matrix of all node hidden representations (graph feature matrix) is indicated

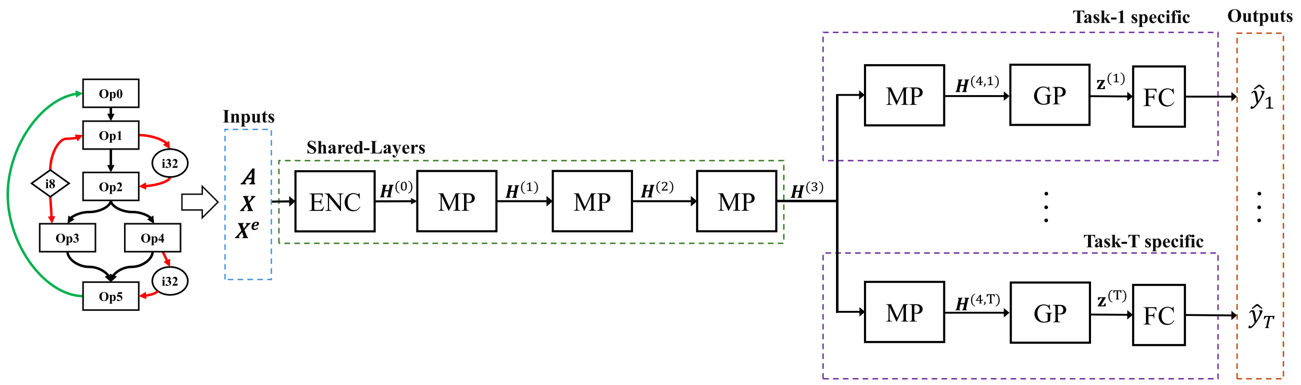


FIGURE 4. Single-input multi-task GNN model ($L = 4$), served as the baseline model in this study.

by $H^{(\ell)} \in \mathbb{R}^{\nu \times c}$, that each node corresponding to a row in the matrix. Additionally, the initial hidden state of nodes is regarded as their initial feature vector, i.e., $h_u^{(0)} = x_u \forall u \in \mathcal{V}$ or, $H^{(0)} = X$. After completing a pre-specified number of iterations (L), the entire graph's representation, denoted by $z \in \mathbb{R}^c$ is obtained by aggregating node embeddings through a readout function Ω_Θ where $z = \Omega_\Theta (h_\upsilon \mid \upsilon \in \mathcal{V})$. This method is called graph pooling (GP), which is performed using simple permutation invariant functions like max/mean/sum or other sophisticated pooling mechanisms [57]. Learning an embedding of the entire graph using this technique in a supervised manner lies principally on the set-based concept [58].

C. SINGLE-INPUT MULTI-TASK GNN-BASED MODEL

This part describes the single-input multi-task GNN model, which is trained and evaluated independently on each variant of the proposed HLS CDFGs data. This model serves as the baseline model in our study.

A multi-task model leveraging hard parameter sharing [59] architecture is utilized to jointly learn multiple tasks. Here, the term "task" refers to a regression task, which involves learning to map inputs to a continuous or numerical output variable. In this case, having T different prediction tasks associated with various HLS resource and performance metrics, the combined objective function of the model is defined as:

$$\mathcal{L}_{\text{total}} = \sum_{\tau=1}^{\text{T}} \mathcal{L}\left(y^\tau, \hat{y^\tau}\right) \tag{4}$$

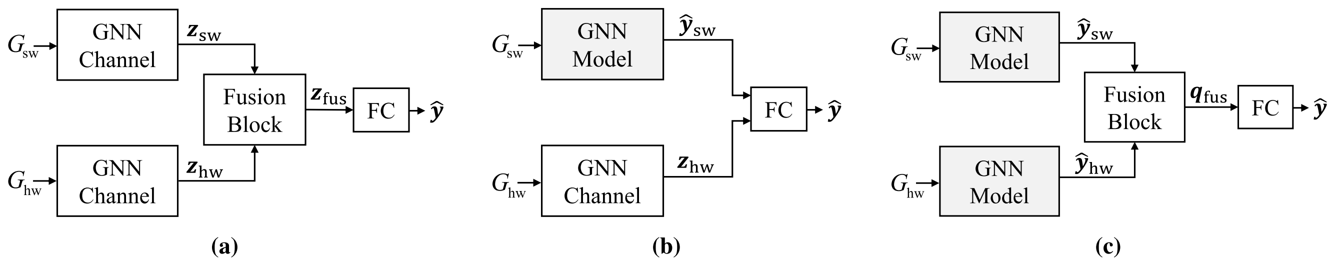
where y^τ denotes a ground truth label for task τ, $\hat{y}^\tau$ is the corresponding model prediction output, and $\mathcal{L}$ is the loss function. The total loss($\mathcal{L}_{total}$) is the sum of the losses from each task.

The overall diagram of the proposed baseline model is depicted in Fig. 4. Node and edge feature matrix along with adjacency matrix serve as the input of the model. Essentially, the proposed GNN model consists of shared and task-specific parts. The shared part begins with an encoder, which is a 3-layer MLP that converts the feature from its initial size to the embedded size. This is followed by some MP layers. Generally, sharing parameters acts as a form of regularization by enforcing the learning of more underlying commonalities that are useful across multiple tasks. Alternatively, task-specific parts have some dedicated layers, tailored to the unique aspects of each task. This allows the model to capture task-specific nuances that shared layers might miss. This part for each task begins with one MP layer followed by a GP and a FC layer. It should be noted that we use superscripts (ℓ) to denote the elements of the shared component, whereas the notation (ℓ, τ) is employed to represent the task-specific components in the rest of the paper.

The performance of GNN models heavily relies on the message-passing algorithm and the number of layers used. Indeed, the MP algorithms define *HOW* information is captured from neighbors, and *HOW* the current state of the node is updated. Having a GNN model with L layers, allows information to be captured from *UP* to L-hop neighbors. Meanwhile, the chosen method must be theoretically sound and capable of generalizing to previously unseen graph data with varying sizes, structures, or features. In our GNN model, graph convolution networks (GCNs) [60] is employed, which serves as a middle ground between spatial and spectral-based GNNs. GCN is a first-order approximation of spectral-based convolutions on graphs. It provides a linear conception of the spectral ChebNet [61] by applying a localized filter (ϑ_θ) on directly connected one-hop neighbors of a node in a graph. The GCN [60] model uses the following spectral convolution formulation:

$$\begin{aligned} X * \vartheta_\theta &= \Theta_0 + \Theta_1 (L - I_N) X \\ &= \Theta_0 X - \Theta_1 D^{\frac{-1}{2}} A D^{\frac{-1}{2}} X \end{aligned} \tag{5}$$

where Θ_0 and Θ_1 are two parameters shared over the graph, $L = I_N - D^{\frac{-1}{2}} A D^{\frac{-1}{2}}$ is the normalized graph Laplacian matrix, I is the identity matrix, and D is the corresponding degree matrix of the adjacency matrix A. However, [60] proposed an improvement to the GCN equation in order to reduce the complexity and computational consumption of the model. Accordingly, for a GNN model consisting of L successive GCN layers, the learned parameters in each



G_{sw}
GNN Channel
$\boldsymbol{z}_{\text{sw}}$
Fusion Block
$\boldsymbol{z}_{\text{fus}}$
FC
$\hat{\boldsymbol{y}}$
G_{hw}
GNN Channel
$\boldsymbol{z}_{\text{hw}}$
(a)
GNN Model
$\hat{\boldsymbol{y}}_{\text{sw}}$
GNN Channel
(b)
$\boldsymbol{q}_{\text{fus}}$
$\hat{\boldsymbol{y}}_{\text{hw}}$
(c)

FIGURE 5. Proposed hybrid GNN models: (a) *jointly-learning and fusion model (JLFM)*, (b) *sequential-learning with knowledge infusion model (SLKIM)*, (c) *parallel learning and fusion model (PLFM)*. In this figure, "GNN Channel" refers to the non-trained model described in III-C, omitting the FC layers, while the GNN model (gray boxes) represents the pre-trained model that provides resource and performance prediction (in inference mode).

layer can be reduced to $\boldsymbol{\Theta}^t = \boldsymbol{\Theta}_0^t = -\boldsymbol{\Theta}_1^t$ as well as normalizing $\boldsymbol{I}_N + \boldsymbol{D}^{\frac{-1}{2}}\boldsymbol{A}\boldsymbol{D}^{\frac{-1}{2}}$ to $\hat{\boldsymbol{D}}$ where $\hat{\boldsymbol{D}}$ is the corresponding diagonal degree matrix used for normalizing the adjacency matrix $\hat{\boldsymbol{A}} = \boldsymbol{A} + \boldsymbol{I}_N$ (self-loop added to each node). The normalization term decreases the instability based on the degree of nodes involved [62]. The final formulation of the hidden representation of nodes is then:

$$\boldsymbol{h}_u^{(\ell+1)} = \sigma\left(\hat{\boldsymbol{D}}^{-\frac{1}{2}}\hat{\boldsymbol{A}}\hat{\boldsymbol{D}}^{-\frac{1}{2}}\boldsymbol{h}_u^{(\ell)}\boldsymbol{W}^{(\ell)}\right) \tag{6}$$

where, $\boldsymbol{W}^{(\ell)}$ is the weight matrix learned during training.

However, we implemented (6) in the form of message-passing network [63] as (7) by considering the edge feature ($\boldsymbol{x}_{u,v}^e$) and adding root embedding, denoted by $\boldsymbol{R}(\boldsymbol{u})$. In this equation, d_i is the degree of node i, $\boldsymbol{W}_e^{(\ell)}$ is the weight matrix to transfer the edge feature, ReLU is rectified linear unit [65] function, and $\omega^{(\ell)} \in \mathbb{R}^1$ is the root embedding weight. $\boldsymbol{R}(u)$ plays an important role in maintaining a direct contribution of the node's original feature to its updated hidden state, independent of the aggregation from its neighbors.

$$\begin{aligned}\boldsymbol{m}_{\mathcal{N}(u)}^{(\ell)} &= \sum_{v\in\mathcal{N}(u)} \frac{1}{\sqrt{\mathrm{d}_v\,\mathrm{d}_u}}\left(\boldsymbol{h}_v^{(\ell)}\boldsymbol{W}^{(\ell)} + \boldsymbol{x}_{u,v}^e\boldsymbol{W}_e^{(\ell)}\right),\\ \boldsymbol{R}(u) &= \mathrm{ReLU}\left(\boldsymbol{h}_u^{(\ell)}\boldsymbol{W}^{(\ell)} + \omega^{(\ell)}\right)\frac{1}{\mathrm{d}_u},\\ \boldsymbol{h}_u^{(\ell+1)} &= \mathrm{ReLU}\left(\boldsymbol{m}_{\mathcal{N}(u)}^{(\ell)}\right) + \boldsymbol{R}(u)\end{aligned} \tag{7}$$

D. HYBRID GNN MODELS

In order to improve the prediction accuracy, three different hybrid GNN models are developed under the multi-task setting by exploiting the two different forms of CDFGs, as described in section III-A. The use of two different CDFG graph representations is intentional and advantageous for several reasons. By incorporating G_{sw}, a consistent structural representation is leveraged that remains unaffected by various HLS optimization directives, allowing the model to maintain a stable framework where pragma information is encoded directly into node features. This stability facilitates pattern recognition across different HLS configurations without structural variability. Conversely, G_{hw} introduces a more dynamic representation, capturing the nuanced structural and feature changes induced by hardware-specific optimizations. This dual-representation strategy provides a comprehensive understanding of both high-level program behavior and low-level hardware-specific characteristics, thereby enhancing the model's ability to generalize across diverse HLS designs. This combined approach allows us to capture a richer set of patterns and dependencies, thus maximizing predictive performance while addressing the limitations inherent in using a single type of graph representation. In the following, a detailed description of the proposed models is provided. It should be noted that the model names were assigned according to the training order.

1) JOINTLY-LEARNING AND FUSION MODEL (JLFM):

In JLFM, each channel leverages the GNN architecture described in Sec. III-C, tough, omitting the FC layers (Fig. 5a). Indeed, each channel plays the role of feature extractor by encoding each input graph data to a set of graph vectors, corresponding to each prediction task (task-specific graph representation vector). Consequently, the model concurrently processes both inputs and optimizes the network to enhance representation by integrating the information from each channel. The final graph representation vector for G_{sw} and G_{hw} inputs is denoted by $z_{\text{sw}}^{(\tau)}$, and $z_{\text{hw}}^{(\tau)}$, where τ is the index of task. Accordingly, for each task ($\tau = 1, 2, 3, \ldots, T$), there exist two different graph representation vectors. The ultimate graph representation in JLFM, represented as $z_{\text{fus}}^{(\tau)}$, is derived by utilizing a fusion block. Despite the variety of fusion techniques available in ML, the sum and weighted-sum fusion are adapted due to their simplicity and effectiveness, making them suitable for our initial exploration of combining software and hardware graph representations.

Summing two vectors involves element-wise addition, resulting in a single vector of the same dimensionality as the input vectors. A weighted sum involves scaling each vector by a weight factor before performing element-wise addition. The associated equations related to the above-mentioned fusion techniques are given in (8) and (9). In these equations, $\oplus$ shows the element-wise addition, and $\omega_{\text{fus}}^{(\tau)} \in \mathbb{R}^1$ is a fusion weight to combine the graph representations of task τ. Indeed, $\omega_{\text{fus}}^{(\tau)}$ is optimized during the training process to achieve an optimal balance between the two vectors and is

TABLE 4. The detailed description of the dataset used in this study.

Benchmark	Design/Function	#Ports	Local Mem.	Seq. Loop	Nes. Loop	$\lvert K_P \rvert$	Explored Pragmas	$\lvert \mathcal{D} \rvert$
AES	aes256_encrypt	3	×	×	✓	4	LP, LUF, AP, FI, DF	2400
BFS	bulk	5	×	×	✓	6	LP, LUF, APF	2256
	queue	5	✓	×	✓	6	LP, LUF, APF	2586
FFT	stride	4	✓	×	✓	10	LP, LUF, APF	1900
GEMM	blocked	3	×	×	✓	11	LP, LUF, LUC, APF	2430
	ncubed	3	×	×	✓	6	LP, LUF, AP	2200
SPMV	ellpack	4	×	×	✓	10	LP, LUF, APF	1868
	crs	5	✓	✓	✓	9	LP, LUF, APF	1450
STENCIL	stencil2d	3	×	×	✓	12	LP, LUF, APF	2380
	stencil3d	3	✓	✓	×	11	LP, LUF, APF	2100
VITERBI	viterbi	5	✓	✓	✓	16	LP, LUF, LUC, APF, APC	1500

Note 1: APF: Array Partitioning Factor, APC: Array Partitioning Complete, LP: Loop Pipelining, LUF: Loop Unrolling Factor, LUC: Loop Unrolling Complete, FI: Function Inlining, DF: Dataflow. **Note 2:** Symbol ✓ signifies the presence of a feature and × indicates its absence.

constrained within the interval [0, 1].

$$z_{fus}^{(\tau)} = z_{sw}^{(\tau)} \oplus z_{hw}^{(\tau)} \tag{8}$$

$$z_{fus}^{(\tau)} = \omega_{fus}^{(\tau)} z_{sw}^{(\tau)} \oplus (1 - \omega_{fus}^{(\tau)}) z_{hw}^{(\tau)} \tag{9}$$

2) SEQUENTIAL-LEARNING WITH KNOWLEDGE INFUSION MODEL (SLKIM)

In SLKIM, a multi-task GNN model (as described in III-C) is initially trained with G_{sw} input to predict relevant metrics. In the subsequent stage, the output predictions from this model are concatenated with the graph representation vector of a GNN channel that encodes G_{hw}. This combined representation is then passed through an FC layer for final prediction. This sequential learning approach effectively infuses knowledge from the initial model into the second stage. The overall diagram of this model is illustrated in Fig. 5b.

3) PARALLEL-LEARNING AND FUSION MODEL (PLFM)

In our third proposed model, each single-input GNN model is trained on its respective input graphs. Then, a fusion model is introduced that receives the prediction results from both models and is trained to learn from them. Indeed, the output of each single-input pre-trained GNN model is a vector with size of T denoted by $\hat{\boldsymbol{y}}_{\mathrm{sw}}$ and $\hat{\boldsymbol{y}}_{\mathrm{hw}}$, corresponding to the models with G_{sw} and G_{hw} inputs. The high-level diagram of the PLFM is depicted in Fig. 5c.

Regarding the fusion of the two vectors, a tensor product (outer product) fusion technique is employed by concatenation of the outputs of the two pre-trained models. This approach allows us to capture the multiplicative interactions between the predictions of the two models, potentially enhancing the model's ability to better learn the highly non-linear relationships between tasks.

$$\boldsymbol{q}_{\mathrm{fus}} = \Gamma(\hat{\boldsymbol{y}}_{\mathrm{sw}} \otimes \hat{\boldsymbol{y}}_{\mathrm{hw}}) || \hat{\boldsymbol{y}}_{\mathrm{sw}} || \hat{\boldsymbol{y}}_{\mathrm{hw}} \tag{10}$$

where $\otimes$ denote the outer multiplication, Γ is the flatten function that transfers a 2D tensor to a vector, and $||$ is the concatenation operation. Consequently, the output $\boldsymbol{q}_{\mathrm{fus}}$ is a vector of size $\mathrm{T}^2 + 2\mathrm{T}$.

IV. EXPERIMENTS AND RESULTS

This section provides a detailed overview of our experiments and describes the evaluation procedures used to assess the accuracy and generalization of the proposed models. To begin with, information on the dataset, environment setup, training setting, and evaluation procedure is included. This is then followed by a comprehensive overview that aims to offer clear insights into the experimental process, demonstrating the robustness and adaptability of our GNN-based HLS DSE framework and supporting its effectiveness with empirical evidence.

A. EXPERIMENTAL SETUP

1) DATASET

Supervised graph learning prediction models require a significant volume of labeled data to achieve effective training and optimization. To fulfill this need, a diverse set of 11 HLS designs from 7 distinct benchmarks, spanning various domains such as encryption algorithms, matrix multiplication, image processing, etc., has been explored. These designs were originally sourced from the MachSuite [51] HLS benchmarks. This was accomplished by applying a diverse range of pragmas, resulting in the generation of hundreds of configurations and corresponding micro-architectures for each HLS design. Table 4 provides a comprehensive overview of the benchmarks utilized, including design/function names, and their high-level characteristics such as the number of ports, the presence of local memory, sequential loops, or nested loops. Additionally, it outlines the number of local knobs ($|K_P|$), a list of pragmas, and design space size ($|\mathcal{D}|$) determined for each design. However, a subset of the design space was successfully implemented, resulting in a total of 12,400 labeled design points, with a minimum of 800 for viterbi and a maximum of 1,500 for AES. It is pertinent to note that our study adhered to a target clock period of 10ns, and the XCZU9EG FPGA from the AMD/Xilinx ZYNQ ULTRASCALE+ family as the target hardware platform.

To extract and represent the software-based CDFGs, the HLS specifications were compiled using Clang [52],

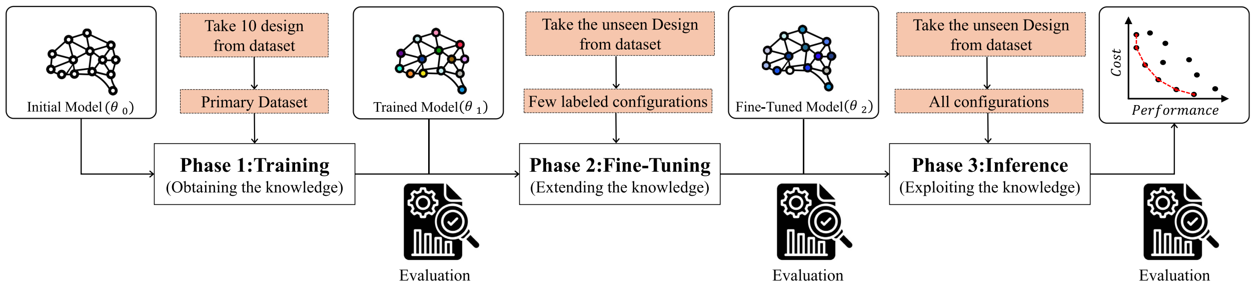

Initial Model (θ_0)
Take 10 design from dataset
Primary Dataset
Phase 1:Training
(Obtaining the knowledge)
Trained Model(θ_1)
Evaluation
Take the unseen Design from dataset
Few labeled configurations
Phase 2:Fine-Tuning
(Extending the knowledge)
Fine-Tuned Model(θ_2)
Evaluation
Take the unseen Design from dataset
All configurations
Phase 3:Inference
(Exploiting the knowledge)
Cost
Performance
Evaluation

FIGURE 6. The QoR prediction and DSE process and evaluation within our GNN-based framework.

a generic and standard C/C++ compiler. The resulting LLVM-based IR was then converted to CDFG format using a modified version of PrograML [64], with pragma information annotated to nodes. Regarding hardware-based CDFGs, VitisHLS [53] is used, as an open-source HLS frontend, which is essentially built upon Clang [52]. The raw graph data is extracted following compilation, transformation, scheduling, and building phases (before RTL generation), and then featured as described in (III-A) using our framework. The *ground truth* values (actual resource usage and performance metrics) were obtained from implementing each configuration using AMD Vivado Design Suite [65].

Due to the substantial difference in the range of labels across different tasks, all ground-truth labels are uniformly scaled using 10Log_{10}, for the sake of ensuring stability during model optimization. This normalization is carried out across all tasks except for CP.

2) TOOLS AND HARDWARE SPECIFICATIONS

Our framework is deployed in Python (v3.8.4) [66]. Meanwhile, PyTorch (v2.0) [67] and PyTorch Geometric (v2.0) ML frameworks [68] are used for implementing the ML models. The dataset is processed on the Lenovo Thinkstation P620 workstation equipped with an AMD Ryzen Threadripper PRO 3945WX CPU and 60 GB RAM running Linux Ubuntu (v22.4) [69]. The training and optimization of the model are performed on the NVIDIA RTX A6000 graphical processing unit (GPU) with 48 GB dedicated memory.

3) EVALUATION PROCEDURE

To evaluate the prediction accuracy of the HLS GNN models discussed in section III and the final HLS DSE Pareto optimal approximation, a rigorous assessment is conducted through a series of distinct experiments. Each experiment is structured into three phases, illustrated in Fig. 6. Accordingly, in the first phase, CDFGs associated with configurations of 10 out of the 11 designs from our dataset are used as the primary graph dataset, while the CDFGs associated with configurations of the remaining design serve as the *unseen* data.

The primary dataset is divided into training, validation, and testing by randomly sampling 70%, 20%, and 10% of the CDFGs of each 10 HLS design, respectively. The training set was used to train the model with initial weights (θ_0), enabling the underlying patterns and relationships within the graph data to be learned. The validation set was employed to monitor the model's performance during training and to adjust hyperparameters accordingly. The test set was utilized to assess the model's performance on unseen configurations from the primary dataset. Once the model acquires the knowledge, it is then employed in the second phase to explore an unseen design, thereby extending its knowledge. To do that, a small portion (10% of the design space in our case) of the unseen design's labeled graph data is exploited to extend the knowledge by fine-tuning the pre-trained model (θ_1). This involves exposing the model to new patterns and intricacies specific to the unseen design, which it had not encountered during the first phase. Subsequently, in the third phase, the fine-tuned model (θ_2) is leveraged to explore and predict the full design space of the unseen design. To ensure a comprehensive evaluation, this procedure is repeated for each of the 11 HLS designs in the dataset. Each design is sequentially designated as unseen and subjected to the same rigorous evaluation process. For each experiment, the models are executed five times, each with a different random seed number, to thoroughly test and validate the model's performance across diverse HLS design scenarios.

4) MODEL AND TRAINING SETTINGS

The hidden dimension of 100 is chosen consistently for all message-passing layers. Regarding the FC layer, an MLP prediction head following a structure of 100-ReLU-200-ReLU-100-1 is used for each prediction task. Considering a batch size of 100 CDFGs, the model is trained in an *inductive setting* [70]. This means the model is trained on one set of graphs and evaluated on entirely unseen graphs, allowing the model to generalize its learned patterns to new, unseen data, making it robust and versatile in practical applications. The Adam gradient-based optimization algorithm [71] was utilized in our model, Known for its efficiency and effectiveness in handling large-scale data and sparse gradients. A learning rate of 1e-2 was used, scheduled to automatically decrease during the training process by a factor of 0.5 with patience of 5, with a maximum of 250 epochs. Also, a weight decay 1e-3 was set. The root mean squared error (RMSE) was employed as the loss function in the multi-task setting, as defined in (4).

TABLE 5. Accuracy of baseline and proposed hybrid models for eight different prediction tasks on the *test set* of the primary HLS graph dataset, measured in MAPE (%) – lower values indicate better performance.

Models		Input(s)	LUT	FF	DSP	CLB	BRAM	CP	CC	T_{exe}	Overall
Baseline	Base-SW	G_{sw}	1.87	2.15	7.38	2.27	4.01	8.50	2.43	2.08	3.84
	Base-HW	G_{hw}	1.64	1.71	7.55	1.99	4.39	6.56	0.95	0.96	3.22
Ours (Hybrid)	JLFM-Sum	$G_{\text{sw}}, G_{\text{hw}}$	1.35	1.46	5.75	1.64	3.87	6.48	1.08	1.06	2.84
	JLFM-WS		1.28	1.34	5.23	1.50	3.5	6.35	0.98	0.96	2.64
	SLKIM	$\hat{\boldsymbol{y}}_{\text{sw}}, G_{\text{hw}}$	1.18	1.26	4.5	1.39	3.41	6.28	0.95	0.96	2.49
	PLFM	$\hat{\boldsymbol{y}}_{\text{sw}}, \hat{\boldsymbol{y}}_{\text{hw}}$	**0.97**	**1.10**	**2.83**	**1.17**	**2.25**	**5.77**	**0.88**	**0.81**	**1.97**

The best-obtained prediction performance of baseline models for each prediction task is indicated with an underline. These for our proposed models are highlighted in bold.

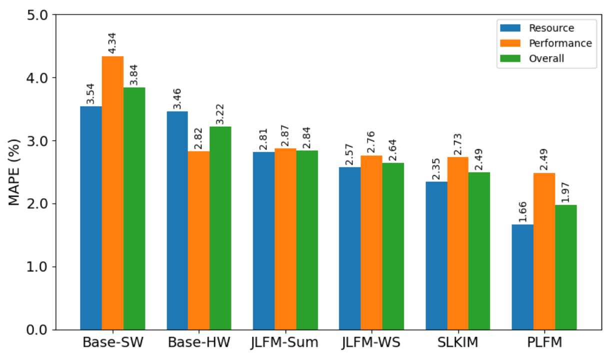


FIGURE 7. Evaluation results on the primary test set.

B. EVALUATION RESULTS ON THE PRIMARY TEST SET

This section presents the evaluation results of both baseline models as well as our proposed hybrid models, each leveraging various input types and architectural configurations detailed in Section (III). The models were assessed across eight different HLS prediction tasks based on the primary test set (at the end of the training phase), providing a thorough comparison of their performance. Table 5 shows the prediction accuracy quantified based on the mean absolute percentage error (MAPE). For each task, the results are averaged over all experiments, as explained in Section (IV-A3). To ensure a fair comparison, uniformity is maintained by employing an identical number of layers ($L = 4$) and adhering to consistent experimental settings across all the compared models. This approach establishes a level playing field, allowing us to specifically isolate the impact of architectural and feature input variations on the performance of the GNN in our application. In this table, Base-SW and Base-HW denote the two baseline models, each employing a multi-task GNN fed by software-based and hardware-based CDFGs, respectively. Additionally, the results of our four hybrid models are presented: JLFM with sum and weighted sum (WS) fusion blocks, SLKIM, and PLFM. Also, Fig. 7 summarizes this table by separately depicting the average prediction accuracy across five resources, three performances, and all eight tasks, overall.

1) EVALUATION RESULTS OF BASELINE MODELS

The comparison between the Base-SW and Base-HW models, evaluated on the test set of the primary dataset, indicates that Base-HW outperforms Base-SW for three out of five HLS resource metrics (LUT, FF, CLB) while showing higher error for DSP and BRAM metrics (Table 5).

When considering the average MAPE across all five resource utilization metrics (Fig. 7), Base-HW demonstrates a slightly better average MAPE of 3.46 compared to 3.54 for Base-SW. Regarding average over three performance metrics (Fig. 7), Base-HW exhibits superior prediction accuracy compared to Base-SW, with an average MAPE of 2.82 versus 4.34 for Base-SW. Base-HW also has a better average MAPE (3.22) compared to Base-SW (3.84), overall. The results highlight that the source and type of input graph can significantly impact the prediction performance of GNN models. Indeed, CDFGs derived from the front end of the HLS compiler, which are optimized for hardware, offer a more effective representation, leading to consistent predictions by GNNs, especially for performance-related HLS metrics.

2) EVALUATION RESULTS OF PROPOSED HYBRID GNN MODELS

Looking at our four proposed hybrid models, evaluated on the primary test set, exhibit significant advancements over the best-performing baseline models for all resource prediction metrics (Table 5). Considering average MAPE values across the five resource prediction tasks (Fig. 7), the JLFM-Sum, JLFM-WS, SLKIM, and PLFM models achieve average MAPE values of 2.81 , 2.57 , 2.35 , and 1.66 , respectively. While the best MAPE among the two baselines is 3.46 (Base-HW). These evaluation results represent substantial relative improvements of 18.79 %, 25.72 %, 32.08 %, and 52.02 % respectively compared to the best-performing baseline. This prediction performance enhancement underscores the effectiveness of our models in more accurately predicting resource utilization metrics.

Regarding performance metric prediction, an improved prediction accuracy for the CP metric is obtained for all proposed models compared to the best of baselines value of 6.56 (Table 5). The improvements are incremental: a slight improvement is shown by JLFM-Sum with a MAPE of 6.48 , followed by JLFM-WS at 6.35 , and SLKIM at 6.28 The most substantial enhancement is achieved by PLFM, with a MAPE of 5.77. Although our hybrid GNN models enhanced the prediction accuracy of CP relative to the baseline, still higher level of error is still seen for CP compared to other metrics. This implies that predicting CP accurately is inherently

TABLE 6. Accuracy of baseline and our two top-performing models for eight different prediction tasks on *unseen designs* after fine-tuning stage using labeled CDFGs. This is performed by using a 10% random sampling from design space of unseen designs, measured in MAPE (%) – lower values indicate better performance.

Models		Input(s)	LUT	FF	DSP	CLB	BRAM	CP	CC	T_{exe}	Overall
Baseline	Base-SW	G_{sw}	$\underline{1.86}$	2.42	9.53	$\underline{2.67}$	$\underline{4.81}$	$\underline{7.05}$	2.51	2.08	4.12
	Base-HW	G_{hw}	2.16	$\underline{2.25}$	$\underline{7.62}$	2.70	2.99	7.46	$\underline{1.23}$	$\underline{1.24}$	3.46
Ours (Hybrid)	SLKIM	$\hat{\boldsymbol{y}}_{\text{sw}}, G_{\text{hw}}$	1.54	1.96	6.44	1.87	2.52	7.01	1.17	1.19	2.96
	PLFM	$\hat{\boldsymbol{y}}_{\text{sw}}, \hat{\boldsymbol{y}}_{\text{hw}}$	**1.17**	**1.44**	**4.87**	**1.36**	**2.38**	**6.20**	**1.07**	**1.06**	**2.44**

The best-obtained prediction performance of baseline models for each prediction task is indicated with an underline. These for our proposed models are highlighted in bold.

more complex due to its interdependencies and non-linearity, coupled with HLS optimizations and the dynamic nature of FPGA routing. Regarding CC and T_{exe}, JLFM-Sum and JLFM-WS achieve results close to the best of baselines. However, a slight degradation is observed. On the other hand, while SLKIM achieved the same value as the best baseline, PLFM surpasses the baseline with a MAPE of 0.88 for CC and 0.81 for T_{exe}, 7.36 % and 15.62 % relative enhancement.

C. COMPLEXITY OF MODELS

Herein, an analysis of the computational cost associated with each model, quantified by the total number of parameters is presented. Understanding the parameter count is crucial for evaluating the trade-offs between model complexity and performance, providing insight into the computational resources required for training and inference. The analysis of model costs reveals distinct differences in parameter counts across the various models. Base-SW and Base-HW have relatively lower parameter counts, at 449 524 and 481 624 , respectively. JLFM-Sum and JLFM-WS exhibit a higher complexity, with 603 124 and 603 132 parameters, respectively. SLKIM presents a moderate increase with 535 512 parameters. In contrast, PLFM significantly exceeds the other models with 4 659 748 parameters, reflecting its increased model complexity. This substantial parameter count for PLFM indicates a higher computational cost, which must be taken into account apart from its superior performance.

D. EVALUATION RESULTS ON UNSEEN HLS GRAPH DATA

In the second phase of the evaluation process, the prediction accuracy of two baseline models along with our top two hybrid GNN models (SLKIM and PTFM) is assessed on *unseen* HLS graph data (unseen designs). Indeed, when a model is trained on the primary dataset (knowledge acquisition phase), it is intended to be applied to other HLS applications and designs that were not part of the training set. Here, we consider a scenario where users have already explored and synthesized a few random samples from the configuration space. These samples can then be utilized to extend the model's knowledge through transfer learning. To do so, the models are fine-tuned using a few labeled data. Knowledge transfer capability is one of the most significant advantages of ML-based HLS DSE compared to analytical model-based and refinement-based methods, enabling to adapt the model to different domains and settings.

Table 6 shows the average MAPE for each task across all unseen designs, and Fig. 8 represents the MAPE based on resource, performance, and overall (average over eight prediction tasks). As is shown in Table 6, the Baseline-SW outperforms Baseline-HW in predicting LUT, CLB, BRAM, and CP metrics, demonstrating a stronger ability to predict these resource and performance characteristics. Conversely, Baseline-HW excels in predicting FF, DSP, CC, and T_{exe}, with particularly notable improvements in CC and T_{exe}. This analysis underscores the varying strengths of each baseline model when it comes to transfer learning, with Baseline-SW being more effective for certain resource metrics, while Baseline-HW offers significant advantages in performance-related predictions. In terms of overall MAPE (Fig. 8), Baseline-HW outperforms Baseline-SW. The overall MAPE for Baseline-HW is 3.46 , while for Baseline-SW, it is 4.12. Baseline-HW achieves a lower MAPE across all tasks, making it the more accurate model overall.

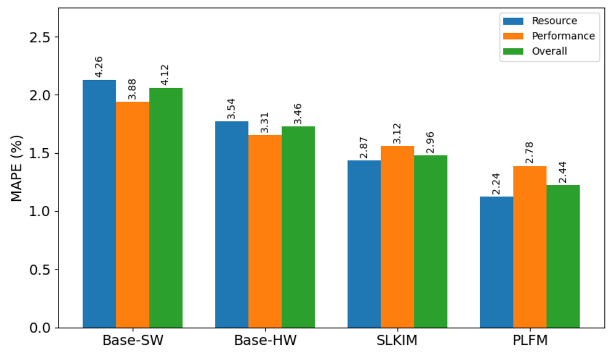


FIGURE 8. Evaluation results of fine-tuning models on unseen CDFG data (from unseen designs).

In contrast, our proposed SLKIM and PLFM models consistently outperform both baselines across all resource prediction tasks (Table 6). This demonstrates the effectiveness of our models in accurately predicting key hardware resources and timings, highlighting their robustness and superior generalization capabilities compared to the baseline approaches. The SLKIM model achieved average MAPE values of 2.87 for resources, 3.12 for performance, and 2.96 overall (Fig. 8). This represents relative improvements of 18.92 %, 5.74 %, and 14.45 %, respectively, compared to the best-performing baselines. Meanwhile, the PLFM

TABLE 7. MAPE (%) of resource and performance prediction reported by HLS tools and SOA frameworks.

Framework/Tool	Type	LUT	FF	DSP	BRAM	CP	CC	T_{exe}
VitisHLS	Intermediate Report	10.4	6.1	7.6	97.0	32.1	**0.0**	2.35
MPSeeker [37]	Analytical Model-Based	13.2	14.7	12.7	19.8	N/A	12.8	N/A
Program-to-Circuit [44]	GNN (HW CDFGs)	2.8	2.9	9.6	3.7	7.6	1.3	1.3
gnn4hls [40]	GNN (SW CDFGs)	2.6	4.8	**1.3**	N/A	N/A	N/A	2.1
Ours	SLKIM	1.5	2.0	6.4	2.5	7.0	1.2	1.2
	PLFM	**1.2**	**1.4**	4.9	**2.4**	**6.2**	1.1	**1.1**

Note 1: In this table, "N/A" signifies instances where data or values were not available. In addition, the top-performing results are highlighted using **bold** for the best values and underline for the second-best values. **Note 2**: To ensure a consistent and accurate comparison, we reproduced the MAPE values for our SLKIM and PLFM models (originally provided in Table 6) by rounding to one decimal place, aligning them with the precision of other works in this comparison.

model demonstrated even greater accuracy, with average MAPE values of 2.24 for resources, 2.78 for performance, and 2.44 overall. These results show the substantial relative improvements of 47.41 %, 16.01 %, and 29.47 % for resources, performance, and overall, respectively, compared to the best-performing baseline models.

These results underscore the effectiveness of our proposed models and highlight the advantage of incorporating fusion techniques to enhance HLS resource and performance prediction accuracy using GNNs. The analysis reveals that prediction accuracy is significantly influenced by the level of knowledge fusion employed. Specifically, given higher levels of knowledge, fusion improves performance, showing that integrating more sophisticated knowledge representations within the GNN framework leads to higher accuracy in predicting HLS metrics.

E. COMPARISON WITH SOA

Comparing our proposed models with SOA tools and frameworks is challenging due to the numerous factors that impact the final prediction accuracy. These factors encompass the benchmarks used, dataset size, dataset splitting, the number of knobs explored and the type of explored pragmas, and the type and features of CDFGs used to represent HLS codes. Additionally, differences in model design and architecture along with experimental settings further complicate direct comparisons. Despite these challenges, here, the VitisHLS report is taken as one of the basis for direct comparison by computing the reported error (in MAPE) at the end of the HLS process for all HLS configurations in our dataset. In addition, the GNN model with hardware CDFG inputs presented in open source Program-to-Circuit [44] framework is taken and re-evaluated based on our dataset. This framework utilized a single-task GNN model (one independent regression model for each prediction objective) with five message-passing layers and a hidden dimension size of 300. For a fair comparison, the GCN [60] is used as the graph convolution algorithm as like as used in our models. On the other hand, the MAPE reported by MPSeeker [37], an analytical model-based tool, and gnn4hls [40], an SOA GNN-based framework are considered for indirect comparison, only by relying on the results provided in their publication based on their data and experiment settings. The latter one (gnn4hls [40]) is constructed based on software-based CDFGs and the model uses the statistical range of applied pragma and their values as graph global features. Moreover, each propagation layer in the proposed GNN model leverages two attention mechanisms. Using a multi-head structure, it lacks task-specific MP layers.

The MAPE for each of the aforementioned models is detailed in Table 7. The CLB metric is excluded from this table, as none of these tools considered this prediction task in their framework. In addition, the MAPE for CC in VitisHLS is shown by zero, reflecting the accurate computation of CC by VitisHLS, which is actually the ground truth for this metric in our dataset.

The data in this table reveals that both of our top-performing hybrid models (SLKIM and PLFM) significantly outperform the VitisHLS tool for all regression tasks, except for CC. also surpass MPSeeker in all available results. When compared to the Program-to-Circuit framework, our SLKIM and PLFM models demonstrate better performance for all regression tasks. This superiority extends to the prediction of LUT, FF, and T_{exe} when compared to gnn4hls, a single-input GNN HLS framework. Our PLFM model estimates LUT, FF, and T_{exe} with MAPE (rounded to one decimal place for standardized comparison) of 1.2 , 1.4 , and 1.1 , respectively, whereas gnn4hls reported 2.6 , 4.8 , and 2.1 MAPE values, respectively.

F. FINE-TUNING TIME EVALUATION

In ML-based solutions for HLS DSE, fine-tuning plays a crucial role by enabling transfer learning and allowing pre-trained models to adapt to new or unseen HLS designs. The fine-tuning time is an important factor, as it directly influences the total DSE runtime. The average fine-tuning times for both baseline models and our two top-performing models were measured on an NVIDIA RTX A6000 GPU (see Fig. 9). To obtain these measurements, each model was run 20 times, using 200 randomly selected labeled CDFGs.

Among baseline, Base-SW demonstrates achieved the quickest fine-tuning time, suggesting it is highly efficient in adapting to new or unseen HLS designs. Its speed makes it particularly advantageous when rapid iteration and real-time adjustments are necessary during the DSE process. Base-HW still offers a relatively fast fine-tuning time. This indicates a balance between efficiency and potentially improved predictive accuracy. The increased time compared

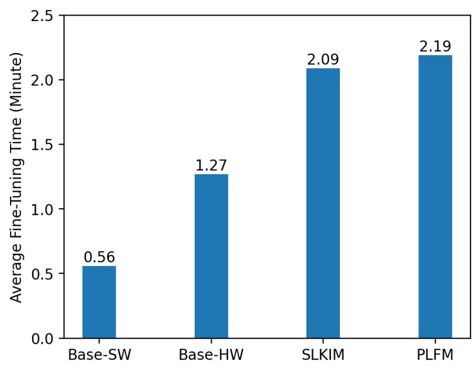


FIGURE 9. The average fine-tuning time (in minutes) for the two baseline models and our two top-performing models, run on the NVIDIA RTX A6000 GPU.

to Base-SW is due to the size and higher complexity of hardware-based CDFGs and feature extraction processes. The proposed SLKIM and PLFM models have longer fine-tuning times due to the higher complexity of models while delivering significantly better accuracy in predictions. The choice between these models should be guided by the specific requirements of the HLS DSE process, balancing the need for speed with the demand for precision.

G. DESIGN SPACE EXPLORATION

The last part of our evaluation aims to assess the fine-tuned models on unseen designs in terms of Pareto optimal approximation. In this regard, the average distance from the reference set (ADRS) figure of merit is used. This metric quantifies the average distance between corresponding points in a reference Pareto point set ($\mathcal{R}$) and approximated Pareto point set ($\mathcal{A}$) by providing a measure of the overall dissimilarity or error between the prediction and the ground truth:

$$\mathrm{ADRS}(\mathcal{R}, \mathcal{A}) := \frac{1}{|\mathcal{R}|} \sum_{r \in \mathcal{R}} \left(\min_{a \in \mathcal{A}} \{\mathrm{dis}\,(r, a)\} \right) \tag{11}$$

where dis( ) is a function that returns the Euclidean distance between two points. Lower ADRS values represent better quality of DSE.

In real-case scenarios, designers choose various pragmas or sets of values for exploration. Thus, taking this into account and ensuring a robust evaluation, we randomly sampled between 40% to 100% of design space for each HLS design and computed the average ADRS. This process iterated 200 times for each DSE assessment, incorporating SLKIM and PLFM as our most effective predictive models, the two GNN-Baseline models, and the VitisHLS tool in our setup. As a case study, we considered the number of LUT utilization as the cost function and the execution time as the performance function. The PF approximation evaluation is given in Table 8.

The results presented in Table 8 demonstrate that both of our proposed models, SLKIM and PLFM, outperform the baseline models (Base-SW and Base-HW) and the VitisHLS tool in terms of Pareto frontier approximation.

TABLE 8. Average ADRS (lower is better) across all designs (f_c = LUT, $f_p = T_{exe}$).

VitisHLS	Base-SW	Base-HW	Ours(SLKIM)	Ours(PLFM)
5.5	3.34	3.06	2.73	2.24

Specifically, the PLFM model achieves the lowest ADRS value of 2.24 , indicating the highest quality of design space exploration. SLKIM follows with an ADRS of 2.73 , showing a moderate improvement over the baseline models. Base-HW and Base-SW have ADRS values of 3.06 and 3.34 , respectively, while VitisHLS lags behind with the highest ADRS of 5.5 In terms of relative improvement, PLFM shows a 26.8 % improvement over Base-HW, a 32.9 % improvement over Base-SW, and a significant 59.3 % improvement over VitisHLS. These results highlight the effectiveness of our models in providing more accurate PF approximations, thereby enhancing the design space exploration process.

V. CONCLUSION

We presented a novel hybrid graph representation and learning framework for enhancing the HLS QoR/DSE problems. Our approach specifically examined the influence of different input graphs on the prediction accuracy of GNN models for various hardware resource and performance prediction tasks, targeting FPGAs. We proposed three distinct GNN-based fusion models that leverage two different forms of CDFGs derived from two different sources, integrated within a multi-task learning architecture.

Our framework demonstrated substantial improvements in both resource utilization and performance estimation compared to single-input baseline models and other SOA methods. Notably, our models achieved better Pareto frontier approximations, enhancing the ability to balance hardware cost and performance trade-offs during the HLS design process. These advancements underscore the potential of our hybrid GNN framework to significantly streamline and optimize HLS DSE, offering more accurate and efficient solutions for FPGA-based system design.

In future research, we plan to explore GNN-based multi-fidelity methods that account for the non-linear relationships present during HLS synthesis, post-synthesis, and post-implementation stages. This approach aims to further enhance prediction accuracy and efficiency by integrating multiple levels of fidelity into the modeling process, ultimately leading to more robust and reliable HLS DSE.

ACKNOWLEDGMENT

The authors would like to thank Canadian Microelectronics Corporation (CMC) Microsystems for providing the design tools.

REFERENCES

[1] B. Peccerillo, M. Mannino, A. Mondelli, and S. Bartolini, “A survey on hardware accelerators: Taxonomy, trends, challenges, and perspectives,” *J. Syst. Archit.*, vol. 129, Aug. 2022, Art. no. 102561.

[2] C. Wang, W. Lou, L. Gong, L. Jin, L. Tan, Y. Hu, X. Li, and X. Zhou, "Reconfigurable hardware accelerators: Opportunities, trends, and challenges," 2017, *arXiv:1712.04771*.

[3] L. Eeckhout, "Is Moore's law slowing down? What's next?" *IEEE Micro*, vol. 37, no. 4, pp. 4–5, Jul. 2017.

[4] H. Esmaeilzadeh, E. Blem, R. St. Amant, K. Sankaralingam, and D. Burger, "Dark silicon and the end of multicore scaling," in *Proc. 38th Annu. Int. Symp. Comput. Archit. (ISCA)*, Jun. 2011, pp. 365–376.

[5] N. Wu, Y. Xie, and C. Hao, "AI-assisted synthesis in next generation EDA: Promises, challenges, and prospects," in *Proc. IEEE 40th Int. Conf. Comput. Design (ICCD)*, Oct. 2022, pp. 207–214.

[6] *IEEE Standard for VHDL Register Transfer Level (RTL) Synthesis*, IEEE Standard 1076.6-2004 (Revision IEEE Standard 1076.6-1999), 2004, pp. 1–118.

[7] *IEEE Standard for Systemverilog–Unified Hardware Design, Specification, and Verification Language*, IEEE Standard 1800-2017 (Revision IEEE Standard 1800-2012), 2018, pp. 1–1315.

[8] D. D. Gajski, N. D. Dutt, A. C. Wu, and S. Y. Lin, *High—Level Synthesis: Introduction to Chip and System Design*. Berlin, Germany: Springer, 2012.

[9] R. S. Molina, V. Gil-Costa, M. L. Crespo, and G. Ramponi, "High-level synthesis hardware design for FPGA-based accelerators: Models, methodologies, and frameworks," *IEEE Access*, vol. 10, pp. 90429–90455, 2022.

[10] R. Nane, V.-M. Sima, C. Pilato, J. Choi, B. Fort, A. Canis, Y. T. Chen, H. Hsiao, S. Brown, F. Ferrandi, J. Anderson, and K. Bertels, "A survey and evaluation of FPGA high-level synthesis tools," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 35, no. 10, pp. 1591–1604, Oct. 2016.

[11] 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, pp. 1–42, 2022.

[12] M. W. Numan, B. J. Phillips, G. S. Puddy, and K. Falkner, "Towards automatic high-level code deployment on reconfigurable platforms: A survey of high-level synthesis tools and toolchains," *IEEE Access*, vol. 8, pp. 174692–174722, 2020.

[13] J. Zhao, L. Feng, S. Sinha, W. Zhang, Y. Liang, and B. He, "Performance modeling and directives optimization for high-level synthesis on FPGA," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 39, no. 7, pp. 1428–1441, Jul. 2020.

[14] J. de Fine Licht, M. Besta, S. Meierhans, and T. Hoefler, "Transformations of high-level synthesis codes for high-performance computing," 2018, *arXiv:1805.08288*.

[15] M. U. Jamal, Z. Li, M. T. Lazarescu, and L. Lavagno, "A graph neural network model for fast and accurate quality of result estimation for high-level synthesis," *IEEE Access*, vol. 11, pp. 85785–85798, 2023.

[16] B. C. Schafer and Z. Wang, "High-level synthesis design space exploration: Past, present, and future," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 39, no. 10, pp. 2628–2639, Oct. 2020.

[17] D. Reyes Fernandez de Bulnes, Y. Maldonado, and L. Trujillo, "Development of multiobjective high-level synthesis for FPGAs," *Sci. Program.*, vol. 2020, no. 1, 2020, Art. no. 7095048.

[18] L. Ferretti, J. Kwon, G. Ansaloni, G. Di Guglielmo, L. P. Carloni, and L. Pozzi, "Leveraging prior knowledge for effective design-space exploration in high-level synthesis," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 39, no. 11, pp. 3736–3747, Nov. 2020.

[19] L. Ferretti, G. Ansaloni, and L. Pozzi, "Cluster-based heuristic for high level synthesis design space exploration," *IEEE Trans. Emerg. Topics Comput.*, vol. 9, no. 1, pp. 35–43, Jan. 2021.

[20] Y.-K. Choi and J. Cong, "HLS-based optimization and design space exploration for applications with variable loop bounds," in *Proc. IEEE/ACM Int. Conf. Comput.-Aided Design (ICCAD)*, Nov. 2018, pp. 1–8.

[21] N. K. Pham, A. K. Singh, A. Kumar, and M. M. A. Khin, "Exploiting loop-array dependencies to accelerate the design space exploration with high level synthesis," in *Proc. Design, Autom. Test Eur. Conf. Exhib. (DATE)*, Mar. 2015, pp. 157–162.

[22] G. Zhong, V. Venkataramani, Y. Liang, T. Mitra, and S. Niar, "Design space exploration of multiple loops on FPGAs using high level synthesis," in *Proc. IEEE 32nd Int. Conf. Comput. Design (ICCD)*, Oct. 2014, pp. 456–463.

[23] G. Zhong, A. Prakash, Y. Liang, T. Mitra, and S. Niar, "Lin-analyzer: A high-level performance analysis tool for FPGA-based accelerators," in *Proc. 53rd ACM/EDAC/IEEE Design Autom. Conf. (DAC)*, Jun. 2016, pp. 1–6.

[24] S. Wang, Y. Liang, and W. Zhang, "FlexCL: An analytical performance model for OpenCL workloads on flexible FPGAs," in *Proc. 54th ACM/EDAC/IEEE Design Autom. Conf. (DAC)*, Jun. 2017, pp. 1–6.

[25] J. Zhao, L. Feng, S. Sinha, W. Zhang, Y. Liang, and B. He, "COMBA: A comprehensive model-based analysis framework for high level synthesis of real applications," in *Proc. IEEE/ACM Int. Conf. Comput.-Aided Design (ICCAD)*, Nov. 2017, pp. 430–437.

[26] M. Motamedi, P. Gysel, V. Akella, and S. Ghiasi, "Design space exploration of FPGA-based deep convolutional neural networks," in *Proc. 21st Asia South Pacific Design Autom. Conf. (ASP-DAC)*, Jan. 2016, pp. 575–580.

[27] Y. Chi, J. Cong, P. Wei, and P. Zhou, "SODA: Stencil with optimized dataflow architecture," in *Proc. IEEE/ACM Int. Conf. Comput.-Aided Design (ICCAD)*, Nov. 2018, pp. 1–8.

[28] L. Ferretti, "Design space exploration in high-level synthesis," Ph.D. thesis, Fac. Inform., Università della Svizzera Italiana, Lugano, Switzerland, 2020.

[29] V. Krishnan and S. Katkoori, "A genetic algorithm for the design space exploration of datapaths during high-level synthesis," *IEEE Trans. Evol. Comput.*, vol. 10, no. 3, pp. 213–229, Jun. 2006.

[30] B. C. Schafer, "Parallel high-level synthesis design space exploration for behavioral ips of exact latencies," *ACM Trans. Des. Automat. Electron. Syst.*, vol. 22, no. 4, pp. 1–20, 2017.

[31] B. Carrion Schafer, "Probabilistic multiknob high-level synthesis design space exploration acceleration," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 35, no. 3, pp. 394–406, Mar. 2016.

[32] S. Kirkpatrick, "Optimization by simulated annealing: Quantitative studies," *J. Stat. Phys.*, vol. 34, no. 5, pp. 975–986, 1984.

[33] V. K. Mishra and A. Sengupta, "PSDSE: Particle swarm driven design space exploration of architecture and unrolling factors for nested loops in high level synthesis," in *Proc. 5th Int. Symp. Electron. Syst. Design*, Dec. 2014, pp. 10–14.

[34] G. Zacharopoulos, A. Barbon, G. Ansaloni, and L. Pozzi, "Machine learning approach for loop unrolling factor prediction in high level synthesis," in *Proc. Int. Conf. High Perform. Comput. Simulation (HPCS)*, Jul. 2018, pp. 91–97.

[35] M. Zuluaga, A. Krause, P. Milder, and M. Püschel, "'Smart' design space sampling to predict Pareto-optimal solutions," in *Proc. 13th ACM SIGPLAN/SIGBED Int. Conf. Lang., Compil., Tools Theory Embedded Syst.*, 2012, pp. 119–128.

[36] J. Kwon and L. P. Carloni, "Transfer learning for design-space exploration with high-level synthesis," in *Proc. ACM/IEEE 2nd Workshop Mach. Learn. CAD (MLCAD)*, Nov. 2020, pp. 163–168.

[37] G. Zhong, A. Prakash, S. Wang, Y. Liang, T. Mitra, and S. Niar, "Design space exploration of FPGA-based accelerators with multi-level parallelism," in *Proc. Design, Autom. Test Eur. Conf. Exhib. (DATE)*, Mar. 2017, pp. 1141–1146.

[38] 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 Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 42, no. 3, pp. 900–913, Mar. 2023.

[39] 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. Automat. Electron. Syst.*, vol. 27, no. 4, pp. 1–27, 2022.

[40] L. Ferretti, A. Cini, G. Zacharopoulos, C. Alippi, and L. Pozzi, "Graph neural networks for high-level synthesis design space exploration," *ACM Trans. Des. Automat. Electron. Syst.*, vol. 28, no. 2, pp. 1–20, 2022.

[41] H. M. Makrani, H. Sayadi, T. Mohsenin, S. Rafatirad, A. Sasan, and H. Homayoun, "XPPE: Cross-platform performance estimation of hardware accelerators using machine learning," in *Proc. 24th Asia South Pacific Design Automat. Conf.*, 2019, pp. 727–732.

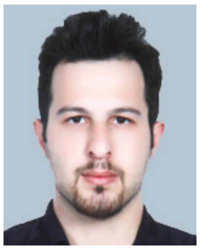
[42] Y. Bai, A. Sohrabizadeh, Y. Sun, and J. Cong, "Improving GNN-based accelerator design automation with meta learning," in *Proc. 59th ACM/IEEE Design Automat. Conf.*, Aug. 2022, pp. 1347–1350.

[43] A. Sohrabizadeh, Y. Bai, Y. Sun, and J. Cong, "Automated accelerator optimization aided by graph neural networks," in *Proc. 59th ACM/IEEE Design Automat. Conf.*, Aug. 2022, pp. 55–60.

[44] N. Wu, H. He, Y. Xie, P. Li, and C. Hao, "Program-to-circuit: Exploiting GNNs for program representation and circuit translation," 2021, *arXiv:2109.06265*.

[45] D. Koch, F. Hannig, and D. Ziener, *FPGAs for Software Programmers*. Cham, Switzerland: Springer, 2016, doi: 10.1007/978-3-319-26408-0.

[46] J. Zhou, G. Cui, S. Hu, Z. Zhang, C. Yang, Z. Liu, L. Wang, C. Li, and M. Sun, "Graph neural networks: A review of methods and applications," *AI Open*, vol. 1, pp. 57–81, Jan. 2020.
[47] Z. Wu, S. Pan, F. Chen, G. Long, C. Zhang, and P. S. Yu, "A comprehensive survey on graph neural networks," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 32, no. 1, pp. 4–24, Jan. 2021.
[48] A. Sohrabizadeh, Y. Bai, Y. Sun, and J. Cong, "Robust GNN-based representation learning for HLS," in *Proc. IEEE/ACM Int. Conf. Comput. Aided Design (ICCAD)*, Oct./Nov. 2023, pp. 1–9.
[49] N. Wu, H. Yang, Y. Xie, P. Li, and C. Hao, "High-level synthesis performance prediction using GNNs: Benchmarking, modeling, and advancing," in *Proc. 59th ACM/IEEE Design Automat. Conf.*, Aug. 2022, pp. 49–54.
[50] N. Wu, J. Lee, Y. Xie, and C. Hao, "LOSTIN: Logic optimization via spatio-temporal information with hybrid graph models," in *Proc. IEEE 33rd Int. Conf. Appl.-Specific Syst., Architectures Processors (ASAP)*, Jul. 2022, pp. 11–18.
[51] B. Reagen, R. Adolf, Y. S. Shao, G.-Y. Wei, and D. Brooks, "MachSuite: Benchmarks for accelerator design and customized architectures," in *Proc. IEEE Int. Symp. Workload Characterization (IISWC)*, Oct. 2014, pp. 110–119.
[52] *Clang: A C Language Family Frontend for LLVM*. [Online]. Available: https://clang.llvm.org/
[53] AMD. (2024). *Xilinx Vitis HLS*. Accessed: May 25, 2024. [Online]. Available: https://github.com/Xilinx/HLS
[54] C. Lattner and V. Adve, "LLVM: A compilation framework for lifelong program analysis & transformation," in *Proc. Int. Symp. Code Gener. Optim. (CGO)*, Mar. 2004, pp. 75–86.
[55] *The LLVM Compiler Infrastructure*. Accessed: Jul. 31, 2024. [Online]. Available: https://www.llvm.org
[56] M. M. Bronstein, J. Bruna, Y. LeCun, A. Szlam, and P. Vandergheynst, "Geometric deep learning: Going beyond Euclidean data," *IEEE Signal Process. Mag.*, vol. 34, no. 4, pp. 18–42, Jul. 2017.
[57] C. Liu, Y. Zhan, J. Wu, C. Li, B. Du, W. Hu, T. Liu, and D. Tao, "Graph pooling for graph neural networks: Progress, challenges, and opportunities," 2022, *arXiv:2204.07321*.
[58] M. Zaheer, S. Kottur, S. Ravanbakhsh, B. Poczos, R. R. Salakhutdinov, and A. J. Smola, "Deep sets," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 30, 2017, pp. 1–11.
[59] S. Vandenhende, S. Georgoulis, M. Proesmans, D. Dai, and L. V. Gool, "Revisiting multi-task learning in the deep learning era," 2020, *arXiv:2004.13379*.
[60] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," 2016, *arXiv:1609.02907*.
[61] M. Defferrard, X. Bresson, and P. Vandergheynst, "Convolutional neural networks on graphs with fast localized spectral filtering," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 29, 2016, pp. 3844–3852.
[62] W. L. Hamilton, "Graph representation learning," *Synth. Lectures Artif. Intell. Mach. Learn.*, vol. 14, no. 3, pp. 1–159, 2020.
[63] J. Gilmer, S. S. Schoenholz, P. F. Riley, O. Vinyals, and G. E. Dahl, "Neural message passing for quantum chemistry," in *Proc. Int. Conf. Mach. Learn.*, 2017, pp. 1263–1272.
[64] C. Cummins, Z. Fisches, T. Ben-Nun, T. Hoefler, M. O'Boyle, and H. Leather, "ProGraML: A graph-based program representation for data flow analysis and compiler optimizations," in *Proc. 38th Int. Conf. Mach. Learn. (ICML)*, Jul. 2021, pp. 2244–2253.
[65] (2022). *Vivado Design Suite User Guide*. [Online]. Available: https://www.xilinx.com/content/dam/xilinx/support/documentation/sw_manuals/xilinx2021_2/ug973-vivado-release-notes-install-license.pdf
[66] *Python*. Accessed: Jun. 25, 2024. [Online]. Available: https://www.python.org/
[67] A. Paszke et al., "PyTorch: An imperative style, high-performance deep learning library," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 32, 2019, pp. 1–12.
[68] M. Fey and J. Eric Lenssen, "Fast graph representation learning with PyTorch geometric," 2019, *arXiv:1903.02428*.
[69] C. Ltd. *Ubuntu—Open Source Operating System*. Accessed: Oct. 22, 2024. [Online]. Available: https://www.ubuntu.com/
[70] W. Hamilton, Z. Ying, and J. Leskovec, "Inductive representation learning on large graphs," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 30, 2017, pp. 1–11.
[71] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," 2014, *arXiv:1412.6980*.



POUYA TAGHIPOUR (Student Member, IEEE) received the B.Sc. and M.Sc. degrees in electronic engineering from Islamic Azad University (IAU), Iran, in 2014 and 2019, respectively. He is currently pursuing the Ph.D. degree with the École de Technologie Supérieure (ÉTS), Montreal, Canada, where he focuses on developing AI/ML-driven methods for the automatic optimization of applications running on FPGA-based heterogeneous computing platforms. His main research interests include FPGA-based hardware acceleration, high-level synthesis (HLS), AI/ML for electronic design automation (EDA), and edge computing.

ERIC GRANGER (Member, IEEE) received the Ph.D. degree in electrical engineering from the École Polytechnique de Montréal, in 2001. He was a Defense Scientist with DRDC Ottawa, from 1999 to 2001, and in research and development with Mitel Networks, from 2001 to 2004. He joined the Department of Systems Engineering, École de Technologie Supérieure (ÉTS), Montreal, Canada, in 2004, where he is currently a Full Professor and the Director of LIVIA, a research laboratory focused on computer vision and artificial intelligence. He is also the FRQS Co-Chair in AI and Health and the ETS Industrial Research Co-Chair on embedded neural networks for intelligent connected buildings (Distech Controls Inc.). His research interests include pattern recognition, machine learning, information fusion, and computer vision, with applications in affective computing, biometrics, face recognition, medical image analysis, and video surveillance.

YVES BLAQUIÈRE (Member, IEEE) received the B.Eng., M.Sc., and Ph.D. degrees in electrical engineering from the École Polytechnique de Montréal, Montreal, QC, Canada, in 1984, 1986, and 1992, respectively. From 1987 to 2016, he was a Professor in microelectronic engineering with the University of Quebec in Montreal (UQAM), Montreal. He has been a Professor with the École de Technologie Supérieure (ÉTS), Montreal, since 2016. He has done research and development projects in collaboration with several microelectronic companies. He is currently a member of the Regroupement Stratégique en Microélectronique du Québec and the Ordre des Ingénieurs du Québec (OIQ). At UQAM, he was the Director of the Laboratoire de Recherche de Conception en Microélectronique, from 1992 to 1999 and from 2004 to 2008, the Director of the Microelectronic Engineering Program, from 2004 to 2010, and the Director of Engineering, from 2011 to 2015. He also works in the field of electrical/electronic/microelectronic engineering, specifically in ASIC/FPGA design, VLSI/WSI microsystems, high-speed digital circuits, timing tools, architectures, defect tolerance, applications in signal processing, and embedded systems.