Learning to Compare Hardware Designs for High-Level Synthesis

Yunsheng Bai[1,2], Atefeh Sohrabizadeh[2], Zijian Ding[2], Rongjian Liang[1], Weikai Li[2], Ding Wang[2], Haoxing Ren[1], Yizhou Sun[2], Jason Cong[2]
[1]NVIDIA Corporation, [2]University of California, Los Angeles
yunshengb@nvidia.com,{atefehsz,bradyd}@cs.ucla.edu,rliang@nvidia.com
weikaili@cs.ucla.edu,allenwang2333@gmail.com
haoxingr@nvidia.com,{yzsun,cong}@cs.ucla.edu

ABSTRACT

High-level synthesis (HLS) is an automated design process that transforms high-level code into optimized hardware designs, enabling rapid development of efficient hardware accelerators for various applications such as image processing, machine learning, and signal processing. To achieve optimal performance, HLS tools rely on pragmas, which are directives inserted into the source code to guide the synthesis process, and these pragmas can have various settings and values that significantly impact the resulting hardware design. State-of-the-art ML-based HLS methods, such as HARP, first train a deep learning model, typically based on graph neural networks (GNNs) applied to graph-based representations of the source code and its pragmas. They then perform design space exploration (DSE) to explore the pragma design space, rank candidate designs using the trained model, and return the top designs as the final designs. However, traditional DSE methods face challenges due to the highly nonlinear relationship between pragma settings and performance metrics, along with complex interactions between pragmas that affect performance in non-obvious ways.

To address these challenges, we propose COMPAREXPLORE, a novel approach that learns to compare hardware designs for effective HLS optimization. COMPAREXPLORE introduces a hybrid loss function that combines pairwise preference learning with pointwise performance prediction, enabling the model to capture both relative preferences and absolute performance values. Moreover, we introduce a novel NODE DIFFERENCE ATTENTION module that focuses on the most informative differences between designs, enhancing the model's ability to identify critical pragmas impacting performance. COMPAREXPLORE adopts a two-stage DSE approach, where a pointwise prediction model is used for the initial design pruning, followed by a pairwise comparison stage for precise performance verification. Experimental results demonstrate that COMPAREXPLORE achieves significant improvements in ranking metrics and generates high-quality HLS results for the selected designs, outperforming the existing state-of-the-art method.

CCS CONCEPTS

• **Hardware → High-level and register-transfer level synthesis**; • **Computing methodologies → Neural networks**.

MLCAD '24, September 9–11, 2024, Salt Lake City, UT, USA

ACM ISBN 979-8-4007-0699-8/24/09
https://doi.org/10.1145/3670474.3685940

KEYWORDS

High-Level Synthesis, Design Space Exploration, Graph Neural Networks, Electronic Design Automation

ACM Reference Format:
Yunsheng Bai[1,2], Atefeh Sohrabizadeh[2], Zijian Ding[2], Rongjian Liang[1], Weikai Li[2], Ding Wang[2], Haoxing Ren[1], Yizhou Sun[2], Jason Cong[2]. 2024. Learning to Compare Hardware Designs for High-Level Synthesis. In *2024 ACM/IEEE International Symposium on Machine Learning for CAD (MLCAD '24), September 9–11, 2024, Salt Lake City, UT, USA.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3670474.3685940

1 INTRODUCTION

High-Level Synthesis (HLS) has emerged as a transformative technology in the realm of hardware design, bridging the gap between high-level software abstractions and efficient hardware implementations. Tools such as Xilinx's Vitis HLS [2] automate the translation of high-level code into optimized hardware designs, enabling rapid development of specialized accelerators for image processing, machine learning, signal processing [6–8, 20], etc.

Central to the HLS design flow is the concept of pragmas—directives embedded within the high-level code that guide the synthesis process, which heavily affects the effectiveness of HLS in producing high-quality designs. However, the relationship between pragma settings and performance metrics is highly nonlinear, with intricate interactions and dependencies that are difficult to predict or reason about. Traditional design space exploration (DSE) methods, which rely on heuristics and iterative synthesis, often fall short in efficiently identifying optimal configurations [41].

To address these challenges, researchers have turned to machine learning (ML) techniques to aid in the DSE process. State-of-the-art ML-based HLS methods, such as GNN-DSE [28] and HARP [29], utilize deep learning models to guide the DSE process for high-quality pragma configurations. These approaches typically involve two key steps: (1) training a predictive model, often based on graph neural networks (GNNs) [15, 39], to learn the mapping between designs with varying pragma settings and performance metrics, and (2) performing DSE using the trained model to rank and select the most promising candidate designs.

While ML-based methods have shown promise in improving the efficiency and effectiveness of HLS, they still face limitations in capturing the complex relationships and interactions within the pragma design space. We hypothesize that the highly nonlinear nature of the design space, coupled with the intricate dependencies between pragmas, poses challenges for accurate performance prediction and the ranking of candidate designs. Our experiments in Section 4 provide further evidence of this issue.

We hypothesize that comparing a pair of designs and predicting which design is better may an easier task compared with accurately predicting the design quality. In this paper, we propose compareXplore, a novel approach to DSE in HLS that leverages the power of comparative learning, a paradigm where models are trained to discern relative preferences between data points [5], to navigate the pragma design space effectively. compareXplore introduces a hybrid loss function that combines pairwise preference learning with pointwise performance prediction, enabling the model to capture both relative preferences and absolute performance values. By learning to compare designs based on their pragma settings and performance characteristics, compareXplore can effectively identify the most promising configurations and guide DSE towards optimal hardware designs.

Moreover, we introduce a novel Node Difference Attention module that focuses on the most informative differences between designs. This attention mechanism allows compareXplore to prioritize the pragma settings that have the greatest impact on performance, enhancing the model's ability to make accurate comparisons and identify critical design choices.

To balance exploration and exploitation in the DSE process, compareXplore uses a two-stage approach. In the first stage, a pointwise prediction model is used to explore and efficiently prune the design space, identifying a subset of promising candidates. This stage leverages the model's ability to estimate absolute performance values and quickly eliminate suboptimal designs. In the second stage, a pairwise comparison model is leveraged to perform precise performance verification and rank the remaining candidates based on their relative performance. This stage takes advantage of the model's comparative learning capabilities to make nuanced distinctions between designs and select the top-performing configurations.

We evaluate compareXplore on a comprehensive set of HLS kernels and demonstrate its effectiveness in improving the quality of the generated hardware designs. Experimental results show that compareXplore achieves significant improvements in ranking metrics, compared to existing state-of-the-art methods. Moreover, the designs selected by compareXplore consistently outperform those obtained through baseline approaches.

The main contributions of this paper are as follows: We propose compareXplore, a two-stage approach to DSE in HLS that leverages comparative learning to effectively navigate the pragma design space and identify high-quality hardware designs.

- We introduce a hybrid loss function that combines pairwise preference learning with pointwise performance prediction, enabling compareXplore to capture both relative preferences and absolute performance values.
- We present a novel Node Difference Attention module that focuses on the most informative differences between designs, enhancing the model's ability to identify critical pragma settings and make accurate comparisons.
- We propose a two-stage DSE approach that balances exploration and exploitation, using a pointwise prediction model for efficient design pruning and a pairwise comparison model for precise performance verification inspired by Ranked Choice Voting (RCV) [1, 23].
- We conduct extensive experiments on a diverse set of HLS benchmarks and demonstrate the superiority of compareXplore over harp in terms of ranking metrics and the quality of the generated hardware designs.

2 RELATED WORK

2.1 ML and GNN for HLS and DSE

Traditional ML algorithms like random forests and linear regression have been used to model HLS tools [19]. However, recent studies demonstrate that GNNs significantly improve accuracy in various HLS tasks [28, 36–38].

Learning algorithms have been applied to accelerate HLS DSE for finding Pareto-optimal designs [36]. In contrast to traditional heuristics-based approaches [30], these methods employ data-driven techniques. IronMan [36], for instance, trains a reinforcement learning agent to optimize resource allocation.

2.2 Pairwise Comparison in ML

Pairwise comparison has a broad range of applications in machine learning beyond its traditional uses in ranking items [21, 26, 35]. In fields such as information retrieval [16] and recommender systems [25], pairwise methods have proven effective for sorting and prioritizing items based on user preferences [5].

Metric Learning In metric learning, pairwise comparisons are used to learn meaningful distances between items. Techniques such as contrastive loss and triplet loss are used to learn a distance metric in which similar items are closer [13, 18].

Preference Learning Pairwise comparison is also central to preference learning [10], where the goal is to learn a model that predicts preferences between items based on observed pairwise comparisons.

Natural Language Processing. Pairwise comparison methods are crucial for tasks such as sentence similarity [32, 34], evaluating machine translation [11], and aligning Large Language Models with human preferences [22, 24, 31].

2.3 Pairwise Comparison in Electronic Design Automation

In the context of hardware optimization, comparative learning and ranking approaches have been explored in the quantum computing domain for optimizing circuit layouts [12], which demonstrates a machine learning based method that ranks logically equivalent quantum circuits based on their expected performance, leading to improvements in reducing noise and increasing fidelity. To the best of our knowledge, we are among the first to adopt the pairwise comparison paradigm in ML-based HLS.

3 METHODOLOGY

3.1 Overview

In this section, we introduce our proposed model, compareXplore, for effective design space exploration in high-level synthesis. compareXplore is a novel comparative learning framework that combines pointwise prediction and pairwise comparison models to efficiently navigate the pragma design space and identify high-quality design configurations.

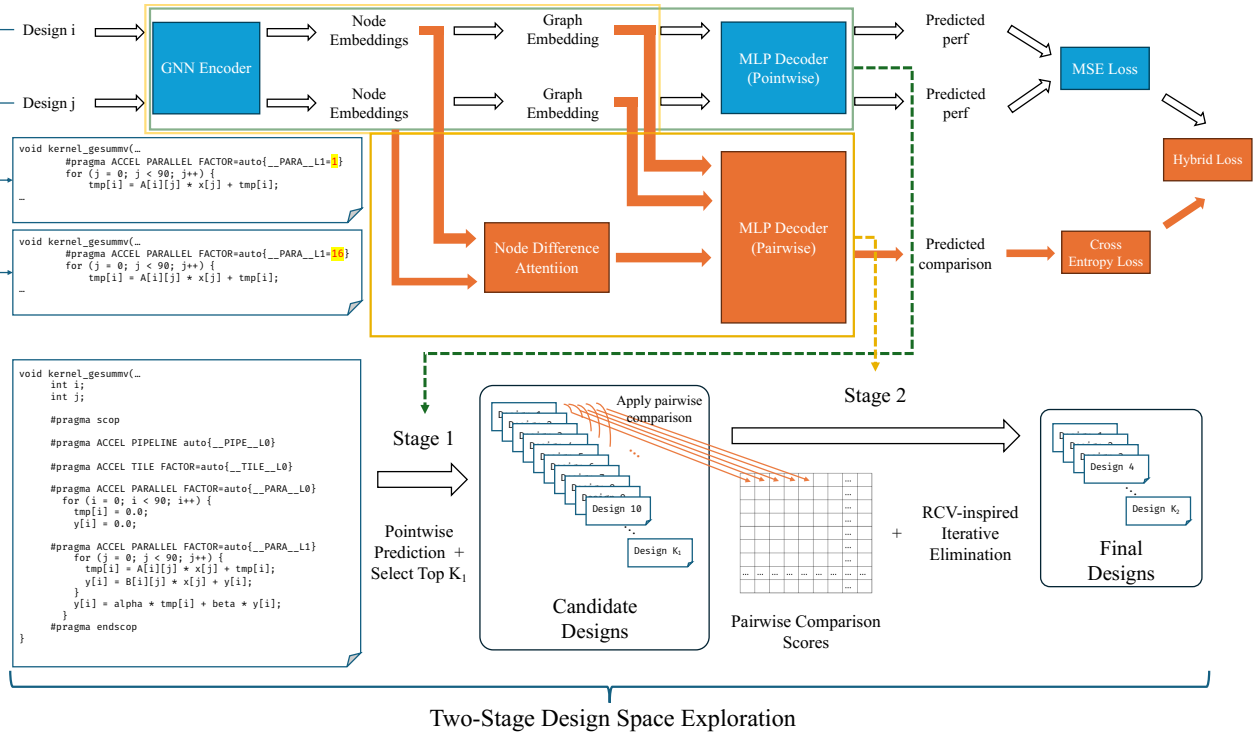

Figure 1: Overview of compareXplore. The model consists of a GNN encoder, a Node Difference Attention module, and two MLP decoders for pairwise comparison and pointwise prediction tasks. The GNN encoder learns node embeddings by aggregating information from neighboring nodes. The Node Difference Attention module focuses on the most informative differences between node embeddings, computes attention scores based on these differences, and aggregates the embedding differences. The model is used in the two-stage DSE process depicted at the bottom. The major novel components are highlighted in the reddish color.

3.2 Problem Setup

Let $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ denote the graph representation of an HLS design used by harp [29], where $\mathcal{V}$ is the set of nodes and $\mathcal{E}$ is the set of edges. Each node $v \in \mathcal{V}$ is associated with a feature vector $\mathbf{x}_v \in \mathbb{R}^d$, where d is the feature dimension. We denote $|\mathcal{G}|$ as the number of nodes.

The goal of design space exploration (DSE) is to find the optimal valid pragma configuration $\mathbf{p}^* \in \mathcal{P}$ that maximizes the performance metric y, which is intuitively the inverse of the latency defined consistently with [29], where $\mathcal{P}$ is the space of all possible pragma configurations.

3.3 Model Architecture

compareXplore consists of a GNN encoder and an MLP-based decoder, as shown in Figure 1.

GNN Encoder: The GNN encoder learns node embeddings by aggregating information from neighboring nodes. We adopt a stack of GNN layers, such as GCN [15], GAT [33], GIN [40], or TransformerConv [27], to capture the structural information and node features of the hardware design graph. The output of the GNN encoder is a set of node embeddings $\mathbf{h}_v \in \mathbb{R}^{d'} | v \in \mathcal{V}$, where d' is the embedding dimension. A pooling operation such as summation is applied to the node embeddings to obtain one embedding per design denoted as $\mathbf{h}_{\mathcal{G}}$.

Node Difference Attention (Node Difference Attention): The Node Difference Attention module is designed to focus on the most informative differences between node embeddings. It takes the node embeddings from the GNN encoder and computes attention scores based on the differences between node pairs. The attention scores are then used to weight the differences, emphasizing the most critical pragma-related differences. The weighted differences are aggregated to obtain a graph-level embedding.

For a design pair (i, j), denote their node embeddings as $\mathbf{H}_i \in \mathbb{R}^{|\mathcal{G}_i| \times d'}$ and $\mathbf{H}_j \in \mathbb{R}^{|\mathcal{G}_j| \times d'}$. Since DSE is only concerned with designs of the same kernel, during training, we only compare designs of the same kernel, i.e. $\mathcal{G}_i$ and $\mathcal{G}_j$ only differ in the pragma nodes and $|\mathcal{G}_i| = |\mathcal{G}_j|$. The differences between the node embeddings in $\mathbf{H}_i$ and $\mathbf{H}_j$ are computed: $\mathbf{D}_{ij} = \mathbf{H}_i - \mathbf{H}_j$. The attention scores are computed by concatenating the node embeddings with their corresponding differences and passing them through an attention network:

$$\mathbf{s}_{ij} = \text{AttentionNet}\big(\text{concat}(\mathbf{H}_i, \mathbf{H}_j, \mathbf{D}_{ij})\big), \quad \mathbf{s}_{ij} \in \mathbb{R}^{|\mathcal{G}_i|}. \tag{1}$$

AttentionNet is a multi-layer perceptron (MLP) that produces attention scores. To focus on the most informative differences, we

propose the following attention mechanism to learn which node-level embedding difference contributes the most to the comparison between the designs:

$$\mathbf{a}_{ij} = \text{softmax}(\mathbf{s}_{ij}), \quad \mathbf{a}_{ij} \in \mathbb{R}^{|\mathcal{G}_i|}. \tag{2}$$

The attention scores are then used to weight the difference embeddings and aggregate the differences:

$$\mathbf{h}_{\mathcal{G}_{ij}} = \sum_{k=1}^{|\mathcal{G}_i|} \text{softmax}(\mathbf{s}_{ij})_k \cdot \mathbf{D}_{ij,k} \tag{3}$$

where $|\mathcal{G}_i| = |\mathcal{G}_j|$ as described previously, and k indicates the k-th element in a $\text{softmax}(\mathbf{s}_{ij})$ and the k-th row in $\mathbf{D}_{ij}$. $\mathbf{h}_{\mathcal{G}_{ij}} \in \mathbb{R}^{d'}$ can be viewed as the graph-level difference-embedding that captures the most informative pragma-related differences.

MLP Decoders: Since there are two tasks, the pairwise design comparison task and the pointwise design prediction task, we use two MLP decoders.

For the pairwise prediction task, the MLP decoder takes the concatenated results of various comparison operations applied to the graph-level embeddings of the two designs as input. Given the graph-level difference-embeddings $\mathbf{h}_{\mathcal{G}_{ij}}$ produced by the Node Difference Attention module, and the individual graph-level embeddings $\mathbf{h}_{\mathcal{G}_i}$ and $\mathbf{h}_{\mathcal{G}_j}$ produced by the pooling operation over $\mathbf{H}_i$ and $\mathbf{H}_j$:

$$\mathbf{h}_{\text{pair}_{ij}} = \text{concat}(\mathbf{h}_{\mathcal{G}_i} \odot \mathbf{h}_{\mathcal{G}_j}, \mathbf{h}_{\mathcal{G}_{ij}}), \tag{4}$$

where $\odot$ denotes the Hadamard product, i.e. element-wise product.

For the pointwise prediction task, the MLP decoder takes the individual design embedding $\mathbf{h}_{\mathcal{G}_i}$ and $\mathbf{h}_{\mathcal{G}_j}$ as input:

$$\mathbf{h}_{\text{point}_i} = \mathbf{h}_{\mathcal{G}_i}, \tag{5}$$

$$\mathbf{h}_{\text{point}_j} = \mathbf{h}_{\mathcal{G}_j}. \tag{6}$$

Both MLP decoders consist of multiple fully connected layers with non-linear activations, such as ReLU. The pairwise MLP decoder outputs a 2-dimensional vector representing the raw logits for the pairwise comparison, while the pointwise MLP decoder outputs a scalar value representing the predicted performance metric for the individual design, i.e.

$$\mathbf{z}_{ij} = \text{MLP}_{pair}(\mathbf{h}_{\text{pair}_{ij}}), \tag{7}$$

$$\mathbf{z}_i = \text{MLP}_{point}(\mathbf{h}_{\text{point}_i}), \tag{8}$$

$$\mathbf{z}_j = \text{MLP}_{point}(\mathbf{h}_{\text{point}_j}). \tag{9}$$

3.4 Training of compareXplore with Hybrid Loss Function

To train compareXplore, we propose a hybrid loss function that combines pairwise preference learning with pointwise performance prediction. This enables the model to capture both relative preferences between designs and absolute performance values.

The hybrid loss function is defined as:

$$\mathcal{L} = \mathcal{L}_{point} + \alpha \mathcal{L}_{pair} \tag{10}$$

where $\alpha \in [0, 1]$ is a hyperparameter that controls the balance between the pairwise and pointwise losses.

The pairwise loss $\mathcal{L}_{pair}$ is calculated using a cross-entropy loss that evaluates the model's ability to correctly rank pairs of design configurations. For each pair of designs, the MLP decoder outputs a 2-D vector of logits, $\mathbf{z}_{ij}$, indicating the model's confidence in the relative performance of designs i and j. The softmax function is applied to these logits to obtain probabilities, $\mathbf{p}_{ij} = [\log(p_{ij}^{(1)}), \log(p_{ij}^{(2)})]$.

The cross-entropy loss for the pairwise comparison is defined as:

$$\mathcal{L}_{pair} = -\sum_{(i,j)\in\mathcal{D}} \Big(\mathbb{1}(y_i > y_j) \log(p_{ij}^{(1)}) + \mathbb{1}(y_i \le y_j) \log(p_{ij}^{(2)}) \Big), \tag{11}$$

where $\mathbb{1}(\cdot)$ is the indicator function, and $\mathcal{D}$ is the set of all pairs (i, j) sampled during training.

The pointwise loss $\mathcal{L}_{point}$ is computed using a mean squared error (MSE) loss between the ground-truth and the predicted performance metric over design pairs sampled for $\mathcal{L}_{pair}$.

3.5 Two-Stage DSE Approach

We propose two-stage approach for design space exploration, as described in Algorithm 1.

Algorithm 1 Two-Stage Design Space Exploration

Stage 1: Use the pointwise prediction model to score and prune the design space to the top K_1 candidate *designs* as done in [29].

Stage 2: Apply pairwise comparisons among the pruned *designs* to obtain $K_2 < K_1$ final designs.

```
scores ← Initialize array of size K1 × K1 with zeros
for i ← 1 to K1 do
    for j ← i + 1 to K1 do
        d_i ← designs[i]
        d_j ← designs[j]
        scores[d_i][d_j] ← PairwiseComparisonModel(d_i, d_j)
    end for
end for
remain ← designs
while |remain| > K2 do
    points ← Initialize array of size |remain| with zeros
    for i ← 1 to |remain| do
        d_i ← remain[i]
        for j ← i + 1 to |remain| do
            d_j ← remain[j]
            points[d_i] ← points[d_i] + scores[d_i][d_j]
            points[d_j] ← points[d_j] + 1 − scores[d_i][d_j]
        end for
    end for
    minPoints ← min(points)
    remain ← remain \ minPoints
end while
return remain
```

In the first stage, the pointwise prediction model is used to efficiently prune the design space $\mathcal{P}$ and identify a subset of K_1 promising candidate designs $\mathcal{P}' \subset \mathcal{P}$. This stage leverages the model's ability to estimate absolute performance values and quickly eliminate suboptimal designs.

Table 1: Main results. The numbers in parentheses indicate the number of designs used for regression (525) and the number of design pairs used for classification (174 for d_1, 827 for d_2, 1411 for d_3, and 8139 for ALL).

Model	Regression (↓)	Classification (↑)				Ranking (↑)
	RMSE (525)	ACC, d_1 (174)	ACC, d_2 (827)	ACC, d_3 (1411)	ACC, ALL (8139)	Kendall's τ
HARP	**0.2333**	**0.8218**	0.8609	0.8866	0.8859	0.4157
COMPAREXPLORE	0.3570	0.8161	**0.8888**	**0.9118**	**0.9117**	**0.4319**

In the second stage, the pairwise comparison model is used to perform precise performance verification on the candidate designs in $\mathcal{P}'$. The pairwise comparisons are used to rank the designs based on their relative performance, and the top-performing designs are selected as the final designs.

Specifically, the design with the minimum total score is iteratively removed from the *remain* set until only K_2 designs are left. This process is analogous to the Ranked Choice Voting (RCV) system, where the candidate with the fewest votes is eliminated in each round. In our case, the pairwise comparison scores serve as the "votes" that determine which designs are eliminated and which ones remain in the top K_2 set. This approach allows for a more nuanced decision-making process.

3.6 Complexity Analysis

Compared to the original HARP model or a pointwise-only approach, the newly introduced NODE DIFFERENCE ATTENTION module has a time complexity of $O(|\mathcal{G}|d')$, which is linear to the number of nodes.

In the two-stage DSE approach, the pointwise prediction stage has the same time complexity as HARP. The newly introduced pairwise comparison stage has a time complexity of $O(K_1^2)$, where K_1 is the number of candidate designs which is usually set to 100. The actual pairwise comparison, i.e. lines 4-10 in Algorithm 1, can be performed in a batch-wise fashion, with a batch size of B. This reduces the time complexity to $O(K_1^2/B)$.

4 EXPERIMENTS

4.1 Experimental Setup

We evaluate our proposed approach using the Xilinx Vitis HLS tool (version 2021.1), which is a more recent version compared to the one used in HARP and the HLSYN benchmark [3]. Our dataset consists of 40 kernels from various application domains. The kernels are synthesized using Vitis HLS, and the resulting designs are used for training and evaluation.

The dataset consists of a total of 10,868 designs, with 9,818 (90.34%) used for training, 525 (4.83%) for validation, and 525 (4.83%) for testing. The test set is used as a transductive test set, where the model has access to the design graphs but not their performance values during training. We ensure all the sampled design pairs come from the training set for a fair comparison. The validation loss is used to select the best model for testing. Training is conducted on a machine with 8 NVIDIA PG506 GPUs.

4.2 Hyperparameter and Implementation Details

Our approach adopts TransformerConv with 7 layers and 64-dimensional node embeddings. Consistent with HARP, we use node attention and encode pragmas using MLPs. The model is trained using the AdamW optimizer with a learning rate of 0.001 and a batch size of 128 for 1600 epochs. We use a cosine learning rate scheduler [17]. The prediction target, performance, is defined consistently with HARP. α is set to 1. For DSE, we set K_1 to 100 and K_2 to 10 with a total time budget of 12 hours with a batch size $B = 512$. The model is implemented in PyTorch Geometric [9]. The full hyperparameters, model implementation, and datasets will be released publicly to enhance reproducibility.

4.3 Evaluation Metrics

We evaluate our approach using two main metrics: pairwise classification accuracy and ranking metrics.

- Pointwise Regression Error: This metric measures the model's ability to accurately make prediction for the performance metric for each design in the test set. The Root Mean Squared Error (RMSE) is used.
- Pairwise Classification Accuracy: This metric measures the model's ability to correctly predict which design in a pair has better performance across design pairs in the test set. We report the accuracy for different degrees of pragma differences (d_1, d_2, d_3 indicating design pairs differing by 1, 2 and 3 pragma settings) and the overall accuracy (ALL).
- Ranking metric: We report Kendall's τ, which measures the ordinal association between the predicted performance rankings and the true performance rankings of the designs in the test[14].

We randomly select six kernels where we perform DSE followed by running the HLS tool to evaluate the selected designs by the DSE process. We report the lowest latency of the selected K_2 designs.

4.4 Loss Curves

Figure 2 presents the training loss curves for our proposed COMPAREXPLORE, including the pairwise loss ($\mathcal{L}_{pair}$), pointwise loss ($\mathcal{L}_{point}$), and the overall loss ($\mathcal{L}$). The figure demonstrates a decrease in both pairwise and pointwise losses, indicating the model's effectiveness in learning from the data for both tasks.

4.5 Main Results

Table 1 presents the comparison between the vanilla HARP model and our proposed approach. Our approach achieves higher pairwise

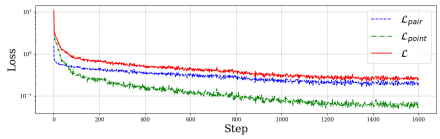

Figure 2: Loss curves for the proposed compareXplore.

classification accuracy for designs with more pragma differences (d_2 and d_3) and overall (ALL) compared to the vanilla HARP model. In terms of ranking performance, our approach achieves a higher Kendall's τ score, indicating a better alignment between the predicted and true rankings of the designs. While HARP achieves a lower regression error, the hybrid loss design in compareXplore leads to a more balanced performance across classification accuracy and ranking metrics.

The accuracy improves as the number of differences between designs increases ($d_1 < d_2 < d_3$). This suggests that the model finds it easier to distinguish between designs that have more differences. When designs differ in more pragmas, the performance metrics tend to vary more significantly, making it easier for the model to learn and identify which design is better. The increasing accuracy from d_1 to d_3 suggests potential future work, such as incorporating curriculum learning to progressively improve the model's performance on more challenging design pairs with smaller performance differences [4].

Figure 3 shows the design space exploration results using our proposed approach compared to the vanilla HARP model. The results demonstrate that our proposed approach consistently outperforms the vanilla HARP model across all kernels in terms of latency reduction. On average, compareXplore achieves a 16.11% reduction in latency compared to HARP. The improvement is particularly significant for the "adi" kernel, where compareXplore reduces the latency by nearly 50%. These results highlight the effectiveness of our approach in identifying high-quality designs that lead to improved hardware performance.

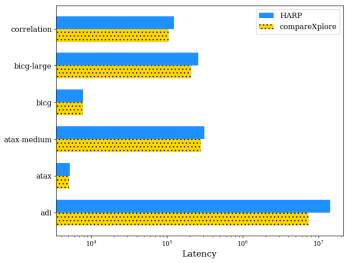

Figure 3: Latency in terms of cycle count (↓) of the final designs selected by the DSE stage. The figure is on the log-scale.

4.6 Parameter Sensitivity Study

Figure 4 shows the effect of α on the compareXplore's performance. As α increases, the regression RMSE worsens, while the classification accuracy peaks around $\alpha = 1$. Kendall's τ ranking metric reaches its highest value at $\alpha = 1$ and then declines. These trends suggest that excessive emphasis on pairwise comparisons may not necessarily improve overall performance. In contrast, moderate α values effectively balance pointwise and pairwise losses, optimizing both tasks effectively.

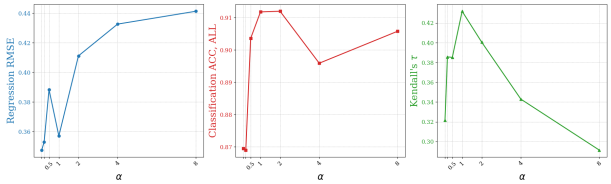

Figure 4: As α increases, the model places more emphasis on the pairwise loss compared to the pointwise loss. α varies in $\{0.125, 0.25, 0.5, 1, 2, 4, 8\}$.

4.7 Time Breakdown Analysis

The average time breakdown analysis presented in Table 2 highlights the efficiency of our two-stage DSE process. On average, Stage 1 accounts for approximately 87.06% of the total computation time, while Stage 2 contributes only 12.94%. This demonstrates that the pairwise comparison phase (Stage 2) introduces minimal additional overhead, ensuring that the overall computational efficiency is maintained. The relatively small proportion of time spent in Stage 2 indicates that our approach is practical and scalable for large-scale design space exploration tasks, making it suitable for optimizing HLS designs.

Table 2: Average Time Breakdown of Stage 1 and Stage 2

Stage	Average Time (%)
Stage 1	87.06
Stage 2	12.94

5 CONCLUSION AND FUTURE WORK

In this paper, we presented compareXplore, a novel approach for HLS design space exploration that addresses the challenges of modeling complex design performance relationships. By incorporating a hybrid loss function, a Node Difference Attention module, and a two-stage DSE approach, compareXplore demonstrates significant improvements in both pairwise comparison accuracy and ranking metrics. Our results show that explicitly learning to compare designs, with a focus on pragma-induced variations, leads to the discovery of higher quality HLS-generated designs.

Although compareXplore does not achieve the lowest regression error compared to HARP, our results show that explicitly learning to compare designs leads to the discovery of higher quality HLS-generated designs. In addition, in practice, it is worth considering a separate model such as HARP for stage 1 of the DSE process specializing in accurate pointwise prediction.

The success of compareXplore in HLS DSE highlights the broader potential of learning-to-rank methods in the hardware optimization domain. In future work, we believe that this paradigm can be further explored and extended. For example, the ability to rank and select designs within a large language model (LLM) framework could lead to tighter integration of language models and hardware design enabling a more intuitive and automated design process, and achieving better performance in both regression and classification due to the higher expressive power of LLMs in capturing complex design relationships and patterns.

REFERENCES

[1] 2023. Ranked Choice Voting. https://www.fairvote.org/rcv. Accessed: May 22, 2024.

[2] AMD. 2024. AMD/Xilinx. https://www.xilinx.com/products/design-tools/vitis/vitis-hls.html. Accessed: June 1, 2024.

[3] Yunsheng Bai, Atefeh Sohrabizadeh, Zongyue Qin, Ziniu Hu, Yizhou Sun, and Jason Cong. 2023. Towards a Comprehensive Benchmark for High-Level Synthesis Targeted to FPGAs. Advances in Neural Information Processing Systems 36 (2023), 45288–45299.

[4] Yoshua Bengio, Jérôme Louradour, Ronan Collobert, and Jason Weston. 2009. Curriculum learning. In Proceedings of the 26th annual international conference on machine learning. 41–48.

[5] Chris Burges, Tal Shaked, Erin Renshaw, Ari Lazier, Matt Deeds, Nicole Hamilton, and Greg Hullender. 2005. Learning to rank using gradient descent. In Proceedings of the 22nd international conference on Machine learning. 89–96.

[6] Andrew Canis, Jongsok Choi, Mark Aldham, Victor Zhang, Ahmed Kammoona, Jason H Anderson, Stephen Brown, and Tomasz Czajkowski. 2011. LegUp: high-level synthesis for FPGA-based processor/accelerator systems. In Proceedings of the 19th ACM/SIGDA international symposium on Field programmable gate arrays. 33–36.

[7] Jason Cong, Jason Lau, Gai Liu, Stephen Neuendorffer, Peichen Pan, Kees Vissers, and Zhiru Zhang. 2022. FPGA HLS today: successes, challenges, and opportunities. ACM Transactions on Reconfigurable Technology and Systems (TRETS) 15, 4 (2022), 1–42.

[8] Jason Cong, Bin Liu, Stephen Neuendorffer, Juanjo Noguera, Kees Vissers, and Zhiru Zhang. 2011. High-level synthesis for FPGAs: From prototyping to deployment. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems 30, 4 (2011), 473–491.

[9] Matthias Fey and Jan E. Lenssen. 2019. Fast Graph Representation Learning with PyTorch Geometric. In ICLR Workshop on Representation Learning on Graphs and Manifolds.

[10] Johannes Fürnkranz and Eyke Hüllermeier. 2010. Preference learning and ranking by pairwise comparison. In Preference learning. Springer, 65–82.

[11] Francisco Guzmán, Shafiq Joty, Lluís Màrquez, and Preslav Nakov. 2019. Pairwise neural machine translation evaluation. arXiv preprint arXiv:1912.03135 (2019).

[12] Gavin S Hartnett, Aaron Barbosa, Pranav S Mundada, Michael Hush, Michael J Biercuk, and Yuval Baum. 2024. Learning to rank quantum circuits for hardware-optimized performance enhancement. arXiv preprint arXiv:2404.06535 (2024).

[13] Elad Hoffer and Nir Ailon. 2015. Deep metric learning using triplet network. In Similarity-Based Pattern Recognition: Third International Workshop, SIMBAD 2015, Copenhagen, Denmark, October 12-14, 2015. Proceedings 3. Springer, 84–92.

[14] Maurice G Kendall. 1938. A new measure of rank correlation. Biometrika 30, 1/2 (1938), 81–93.

[15] Thomas N Kipf and Max Welling. 2016. Semi-supervised classification with graph convolutional networks. arXiv preprint arXiv:1609.02907 (2016).

[16] Tie-Yan Liu et al. 2009. Learning to rank for information retrieval. Foundations and Trends® in Information Retrieval 3, 3 (2009), 225–331.

[17] Ilya Loshchilov and Frank Hutter. 2016. Sgdr: Stochastic gradient descent with warm restarts. arXiv preprint arXiv:1608.03983 (2016).

[18] Marc Masana, Idoia Ruiz, Joan Serrat, Joost van de Weijer, and Antonio M Lopez. 2018. Metric learning for novelty and anomaly detection. arXiv preprint arXiv:1808.05492 (2018).

[19] Pingfan Meng, Alric Althoff, Quentin Gautier, and Ryan Kastner. 2016. Adaptive threshold non-Pareto elimination: Re-thinking machine learning for system level design space exploration on FPGAs. In 2016 Design, Automation & Test in Europe Conference & Exhibition (DATE). IEEE, 918–923.

[20] Razvan Nane, Vlad-Mihai Sima, Christian Pilato, Jongsok Choi, Blair Fort, Andrew Canis, Yu Ting Chen, Hsuan Hsiao, Stephen Brown, Fabrizio Ferrandi, et al. 2015. A survey and evaluation of FPGA high-level synthesis tools. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems 35, 10 (2015), 1591–1604.

[21] Sahand Negahban, Sewoong Oh, and Devavrat Shah. 2012. Iterative ranking from pair-wise comparisons. Advances in neural information processing systems 25 (2012).

[22] Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, et al. 2022. Training language models to follow instructions with human feedback. Advances in neural information processing systems 35 (2022), 27730–27744.

[23] Richard H Pildes and Michael Parsons. 2021. The Legality of Ranked-Choice Voting. Cal. L. Rev. 109 (2021), 1773.

[24] Rafael Rafailov, Archit Sharma, Eric Mitchell, Christopher D Manning, Stefano Ermon, and Chelsea Finn. 2023. Direct preference optimization: Your language model is secretly a reward model. Advances in Neural Information Processing Systems 36 (2023).

[25] Lior Rokach and Slava Kisilevich. 2012. Initial profile generation in recommender systems using pairwise comparison. IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews) 42, 6 (2012), 1854–1859.

[26] Nihar B Shah and Martin J Wainwright. 2018. Simple, robust and optimal ranking from pairwise comparisons. Journal of machine learning research 18, 199 (2018), 1–38.

[27] Yunsheng Shi, Zhengjie Huang, Shikun Feng, Hui Zhong, Wenjin Wang, and Yu Sun. 2021. Masked label prediction: Unified message passing model for semi-supervised classification. ICAI (2021).

[28] Atefeh Sohrabizadeh, Yunsheng Bai, Yizhou Sun, and Jason Cong. 2022. Automated accelerator optimization aided by graph neural networks. In Proceedings of the 59th ACM/IEEE Design Automation Conference. 55–60.

[29] Atefeh Sohrabizadeh, Yunsheng Bai, Yizhou Sun, and Jason Cong. 2023. Robust GNN-based representation learning for HLS. In 2023 IEEE/ACM International Conference on Computer Aided Design (ICCAD). IEEE, 1–9.

[30] Atefeh Sohrabizadeh, Cody Hao Yu, Min Gao, and Jason Cong. 2022. AutoDSE: Enabling software programmers to design efficient FPGA accelerators. ACM Transactions on Design Automation of Electronic Systems (TODAES) 27, 4 (2022), 1–27.

[31] Feifan Song, Bowen Yu, Minghao Li, Haiyang Yu, Fei Huang, Yongbin Li, and Houfeng Wang. 2024. Preference ranking optimization for human alignment. In Proceedings of the AAAI Conference on Artificial Intelligence, Vol. 38. 18990–18998.

[32] Xiaofei Sun, Yuxian Meng, Xiang Ao, Fei Wu, Tianwei Zhang, Jiwei Li, and Chun Fan. 2022. Sentence similarity based on contexts. Transactions of the Association for Computational Linguistics 10 (2022), 573–588.

[33] Petar Velickovic, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, Yoshua Bengio, et al. 2017. Graph attention networks. stat 1050, 20 (2017), 10–48550.

[34] Hongwei Wang and Dong Yu. 2023. Going beyond sentence embeddings: A token-level matching algorithm for calculating semantic textual similarity. In Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers). 563–570.

[35] Fabian Wauthier, Michael Jordan, and Nebojsa Jojic. 2013. Efficient ranking from pairwise comparisons. In International Conference on Machine Learning. PMLR, 109–117.

[36] Nan Wu, Yuan Xie, and Cong Hao. 2021. Ironman: Gnn-assisted design space exploration in high-level synthesis via reinforcement learning. In Proceedings of the 2021 on Great Lakes Symposium on VLSI. 39–44.

[37] Nan Wu, Yuan Xie, and Cong Hao. 2022. IRONMAN-PRO: Multiobjective design space exploration in HLS via reinforcement learning and graph neural network-based modeling. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems 42, 3 (2022), 900–913.

[38] Nan Wu, Hang Yang, Yuan Xie, Pan Li, and Cong Hao. 2022. High-level synthesis performance prediction using gnns: Benchmarking, modeling, and advancing. In Proceedings of the 59th ACM/IEEE Design Automation Conference. 49–54.

[39] Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and S Yu Philip. 2020. A comprehensive survey on graph neural networks. IEEE transactions on neural networks and learning systems 32, 1 (2020), 4–24.

[40] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. 2019. How powerful are graph neural networks? ICLR (2019).

[41] Wei Zuo, Yun Liang, Peng Li, Kyle Rupnow, Deming Chen, and Jason Cong. 2013. Improving high level synthesis optimization opportunity through polyhedral transformations. In Proceedings of the ACM/SIGDA international symposium on Field programmable gate arrays. 9–18.

Received 29 July 2024; revised 29 July 2024; accepted 14 Aug 2024