Efficient Task Transfer for HLS DSE

Zijian Ding, Atefeh Sohrabizadeh, Weikai Li, Zongyue Qin, Yizhou Sun, Jason Cong

{bradyd,atefehsz,weikaili,qinzongyue,yzsun,cong}@cs.ucla.edu

University of California, Los Angeles

Los Angeles, CA, USA

ABSTRACT

There have been several recent works proposed to utilize model-based optimization methods to improve the productivity of using high-level synthesis (HLS) to design domain-specific architectures. They would replace the time-consuming performance estimation or simulation of design with a proxy model, and automatically insert pragmas to guide hardware optimizations. In this work, we address the challenges associated with high-level synthesis (HLS) design space exploration (DSE) through the evolving landscape of HLS tools. As these tools develop, the quality of results (QoR) from synthesis can vary significantly, complicating the maintenance of optimal design strategies across different toolchains. We introduce Active-CEM, a task transfer learning scheme that leverages a model-based explorer designed to adapt efficiently to changes in toolchains. This approach optimizes sample efficiency by identifying high-quality design configurations under a new toolchain without requiring extensive re-evaluation. We further refine our methodology by incorporating toolchain-invariant modeling. This allows us to predict QoR changes more accurately despite shifts in the black-box implementation of the toolchains. Experiment results on the HLSyn benchmark transitioning to new toolchain show an average performance improvement of 2.38× compared to AutoDSE and a 1.2× improvement over HARP, while also increasing the sample efficiency by 5.75×, and reducing the runtime by 2.7×.

CCS CONCEPTS

• **Hardware → Electronic design automation**; • **Computing methodologies → Machine learning**.

KEYWORDS

high-level syntheis, transfer learning, optimization

1 INTRODUCTION

General-purpose computers are widely employed across diverse domains owing to their ease of programming. However, they face significant overheads due to extended instruction pipelines necessary for supporting generalization. This has given rise to domain-specific
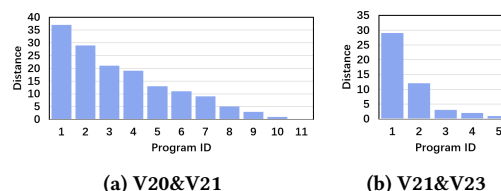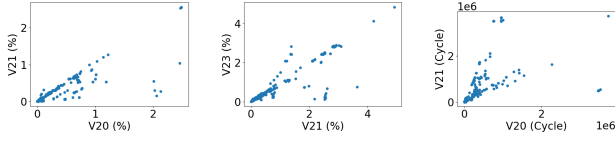
(a) V20&V21   (b) V21&V23

Figure 1: The distance of the best design found by AutoDSE [28] between different toolchains: The horizontal axis represents different programs, and the vertical axis denotes the distance between designs. V20: Vitis HLS 2020.2, V21: Vitis HLS 2021.1, V23: Vitis HLS 2023.2. The distance is calculated by summing up the discrete code difference between two designs over each pragma.

accelerators (DSAs) [8, 9, 14]. However, DSAs pose challenges in programming, restricting their user base to hardware experts [8]. High-level synthesis (HLS) [11] was introduced to simplify the design process by raising the abstraction level from the register-transfer level (RTL) to C/C++. Designers use compiler directives, in the form of pragmas, to describe microarchitecture. While HLS reduces design turn-around times, not every HLS design yields a high-performance microarchitecture [28]. As a result, a new line of research aims to explore the solution space defined by pragmas more efficiently. Because evaluating each design candidate with HLS tools is time-consuming, recent studies propose supervised learning models as proxies of HLS tools, which can predict the quality of results (QoR) to expedite exploration [6, 26, 30–32]. Training such models involves collecting extensive datasets of diverse designs synthesized by HLS tools.

A significant issue arises from the continuous evolution of HLS tools, which can significantly impact QoR values [27]. To gain insight into this issue, we utilized AutoDSE [28] to gather data on different Vitis HLS toolchains: 2020.2 (V20) [1], 2021.1 (V21) [2] and 2023.2 (V23) [3]. Figure 2 illustrates how the performance and resource utilization of each design may vary when changing the toolchain. Additionally, as shown in Table 1, the validity of a design may also fluctuate. During HLS design synthesis, encountering errors is common. For instance, the HLS tool may refuse to synthesize a design with high parallelization factors [26], or it may timeout during synthesis. It is worth noting that we only consider designs within the intersection of those explored by AutoDSE, and more significant changes may occur when evaluating designs across the entire design space. One question that arises is whether changes in performance/resource labels and validity lead to alterations in the optimal design scheme. For instance, a design that previously performed well with one toolchain may become invalid with another. To understand how changes in HLS tools affect the design scheme, we run AutoDSE on these toolchains and compare the best design

| (a) LUT-V20&V21 | (b) LUT-V21&V23 | (c) Latency-V20&V21 |

Figure 2: The label shift when changing the toolchain: the performance/resource labels are plotted on the X/Y axis for the old/new toolchain, a larger distance from the $Y = X$ line indicate larger label shift.

Table 1: The validity changes when altering the toolchain. We compare two pairs of commercial HLS tools: V20&V21, V21&V23. "V1" and "V0" stand for the valid or invalid design in the corresponding toolchain.

(a) V20&V21

	V21 V1	V21 V0
V20 V1	1462	295
V20 V0	269	348

(b) V21&V23

	V23 V1	V23 V0
V21 V1	3690	120
V21 V0	72	1814

it finds. AutoDSE optimizes design performance based on feedback from the toolchain and can naturally adapt to new toolchains. Therefore, any change in the best design found by AutoDSE is attributed to the change in the downstream toolchain. Upon applying AutoDSE to the new toolchain, we observed an average performance improvement of 2×. Furthermore, we noted that the change in the performance of the best design arises from the modification in the design scheme. Essentially, AutoDSE can discover new and improved designs on the new toolchain. Figure 1 depicts the editing distance of the best designs found by AutoDSE on the new and old toolchains. The pragma edit distance is calculated by summing up the differences between each pragma. For example, if one pragma P_1 can take values from $[1, 2, 5]$, and one design takes value 5 while the other takes value 1, then their discretized values (index in the design space) are 3 and 1 respectively. This results in a contribution of $3 - 1 = 2$ to the total editing distance. A larger editing distance indicates a greater shift in the design scheme.

It is therefore critical to design a task transfer learning scheme that could efficiently adapt to the evolving toolchain. According to the previous observations, we observe two major challenges. (1) The first one is to sample efficiently the design from the design space so that the model transferred to those designs can effectively identify the high-quality designs. (2) Another challenge is to adapt the model to the large label shift. Overall, we want to efficiently uncover good designs on the new toolchain when the label shift is unknown and the design space could be as large as 10^{13} as we see in the HLSyn benchmark [5].

Model-based optimization is a widely used technique for efficiently sampling high-quality designs. Previous work has utilized Bayesian Optimization [7, 29], where sampling is guided by an acquisition function. However, optimizing the acquisition function in a large discrete space, such as HLS designs, remains challenging. Other studies, for example [20] have adapted active learning to achieve better sample efficiency when transferring to a new technology node by proposing data selection algorithms that select diverse and informative data from existing datasets. Our approach is not limited to selecting data from an existing unlabeled data pool,

and could navigate the whole design space. In [26, 27, 30–32], optimizing HLS designs involves training a proxy model on an offline dataset and then conducting optimization with this proxy model. This offline approach requires time-consuming data collection and suffers from generalizability issues when applied to new domains and tasks.

Transfer learning investigates the generalizability of models to unseen domains and tasks, with representation learning playing a critical role in domain transferability [6, 13, 17, 33–36]. Some studies propose learned design space pruning for domain transfer in Bayesian Optimization [4, 23]. Unlike domain transfer, which assumes a data distribution shift, we focus on task transfer, assuming a tractable change in the black-box function. Previous works [20, 27] have shown better task transfer accuracy by freezing some model parameters.

To enable efficient sampling on an unseen toolchain, we propose a novel model-based explorer designed for discrete optimization spaces. We jointly optimize the model prediction and the importance sampling distribution, allowing the sampling process to leverage knowledge from the previous toolchain and achieve better sample efficiency. Moreover, the sampling distribution can guide model updates, enabling the model to minimize its error on a subset of the design space rather than striving for perfect accuracy across the entire space. We also explore learning task-generalizable representations of HLS designs. Our key observation is that the same input often represents similar micro-architectures, despite potential differences in their implementations across toolchains. Thus, we aim to learn representations of HLS designs that can extract their invariance across various toolchains.

Overall, we make the following contributions:

- We propose a model-based explorer that efficiently searches the whole design space, with better sample efficiency compared with the SoTA approach.
- We introduce a novel modeling of HLS designs to capture the invariance between toolchains.
- Evaluation on 13 programs transferring to V21 shows that we can improve design performance by 2.38× and 1.2× on average, compared with AutoDSE and HARP, with a sample efficiency of 5.75× and a 2.7× runtime improvement.
- Evaluation on the latest toolchain reveals that our design outperforms the best of AutoDSE and HARP by 1.27×.

2 PRELIMINARIES

2.1 HLS Design Space and Pragmas

As with AutoDSE and HARP, we utilize the open-source AMD/Xilinx Merlin Compiler [10] to streamline DSE for HLS. It supports three types of pragmas: PIPELINE, PARALLEL, and TILE. Taking designs with these pragmas as input, it automates the synthesis of on-chip buffers and memory interfaces. The PIPELINE, PARALLEL, and TILE pragmas handle loop pipelining, double buffering, loop unrolling, and loop tiling for parallelization and latency hiding. The Merlin Compiler supports standard pipelining and parallelization of HLS, but also offers additional automation for pragmas like "array_partition" which can be optimally solved given the parallel and tiling factors. This compiler provides a more compact design space and is used in this work. However, our approach can be directly applied to other HLS tools as well.
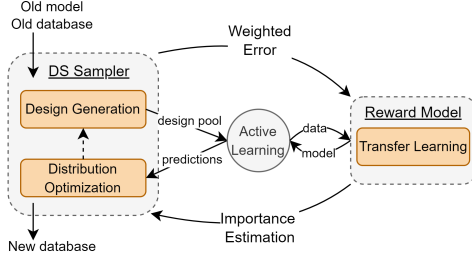
Figure 3: Overview

2.2 HARP

The HARP framework [27] is the state-of-the-art model-based approach for predicting the performance and resource utilization of HLS designs. It introduces a hierarchical graph representation to mitigate the over-smoothing issue encountered in graph neural networks. By employing a hierarchical graph, the shortest path of the graph can be substantially reduced. This enables the capture of more global information with a shallow GNN. Furthermore, HARP proposes to model the program and pragma transformations separately, treating pragmas as transformations modeled by Multi-Layer Perceptrons (MLPs). This model design aligns with the nature of HLS designs and results in improved accuracy and DSE performance. It trains a classification model to predict the validity and a regression model to predict the QoR. We follow this convention and denote the classification model as $\hat{R}_{\theta_c}$ and the regression model as $\hat{R}_{\theta_r}$.

3 METHODOLOGY

Our primary objective is for the learned model to adapt to a new toolchain while identifying Pareto-optimal points and maximizing sample efficiency. Our approach focuses on employing importance sampling to select high-quality designs and integrating model prediction within the importance sampling algorithm to improve sample efficiency. Figure 3 illustrates an overview of our proposed method, which consists of two main components: the DS Sampler and Reward Model Transfer Learning. Using the Cross-Entropy Method (CEM), the DS Sampler performs importance sampling based on the design performance. It iteratively optimizes a probability distribution across the entire design space, concentrating higher density on designs with high rewards. This approach ensures that high-reward designs are sampled more frequently, thereby increasing the likelihood of identifying optimal designs. To further boost sample efficiency, we introduce a reward model that predicts the validity and QoR of HLS designs. This model is adapted from previous toolchains, leveraging data collected from earlier implementations to enhance prediction accuracy. To address the potential label shift due to toolchain changes and distribution shift from the importance sampling, we integrate an active learning subroutine within the CEM algorithm. In summary, we facilitate an efficient transfer to a new toolchain through a model-based method, which involves two iterative steps: fitting the model to the current sampling distribution and updating the sampling distribution based on the model's recommendations.

In Section 3.1, we will describe version-invariant modeling that efficiently adapts the model to the label shift. In Section 4, we will explain the model-based importance sampling algorithm based on CEM.
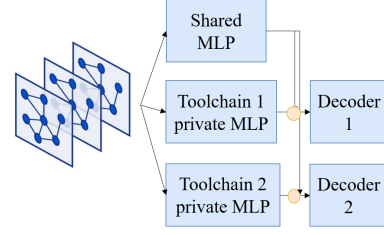

Figure 4: Model architecture with invariant embedding

3.1 Toolchain-invariant modeling

A critical problem of training a prediction model for HLS designs is how to train a robust model when the labeled data is scarce compared to the size of the design space. One way to mitigate the insufficiency of labeled data is to transfer knowledge from all existing data, possibly collected from different toolchains. Our intuition is to model a robust representation of HLS designs that contains the invariant information between different toolchains. In practice, such invariance is abundant. For example, the same input program with the same pragmas represents similar microarchitectures, and similar microarchitectures will not result in completely different implementations. Following this intuition, we consider learning invariant embedding for task transfer on the HLS design by training a single model with data from different toolchains.

One problem to tackle is to train a model that is aware of the version of the data. A simple solution is to include the version tag in the input data. However, when switching to a new version, we may need a completely new tag, and that will cause a huge shift in input data. Another approach is to have a shared encoder, but different decoder for each task. Each decoder is trained on data from its corresponding toolchain, allowing the unique parameters of the decoders to capture the specific characteristics of each toolchain. This approach follows the intuition that the same input HLS design represents a similar micro-architecture for different toolchains, and could be represented as the same embedding in the latent space. However, we argue that it is not enough to just model the shared information between toolchains. A shared embedding may not contain enough information to address the variance between toolchains. In HLS design, the information that remains completely invariant is the program. But for pragma transformations, it remains unknown exactly what implementation stays the same and what is changed. Following this intuition, we propose the following modeling scheme:

Separate embedding for shared and private information. Fig. 4 shows the model architecture. For each input design, two separate embeddings of the design will be learned. One of the embeddings is modeled by a set of parameters shared among all toolchains, and the other embedding is modeled with a set of parameters that is private to each toolchain. Then, we concatenate the two embeddings and forward it to different decoders for each toolchain.

Another benefit of the proposed model architecture is that it can be trained jointly with data from multiple versions. We optimize the mean-squared error (MSE) averaged over all versions of data. Intuitively, training with more data will help the model get a stronger representation of the invariant part. Moreover, once diverse data from different toolchains have been gathered, the invariant part

Algorithm 1 Cross-entropy method for optimization (model-free)

1: **Input:** distribution hypothesis set Θ, number of iterations n, sample per iteration N, true function R, quantile ρ, step size α
2: **Output:** max reward r_{max}
3: Initialize sampling distribution $p(x)$ to random.
4: Initialize maximum reward to $r_{max} = -\infty$
5: **for** $t = 1$ **to** n **do**
6:     $\mathcal{D}_s^t \leftarrow N$ samples without replacement from $p(x)$
7:     $\mathcal{R}_s^t \leftarrow$ Evaluate $R(x)$ for $x \in \mathcal{D}_s^t$.
8:     $r_{max} = max(r_{max}, max(r \in \mathcal{R}_s^t))$
9:     $q \leftarrow$ Calculate the $1 - \rho$ quantile of $\mathcal{R}_s^t$
10:    $\hat{p}(x) = arg\,max_{p \in \Theta} \sum_{x \in \mathcal{D}_s^t} p(x) \mathbb{1}[R(x) > q]$
11:    $p(x) = (1 - \alpha)p(x) + \alpha\hat{p}(x)$
12: **end for**
13: **return** r_{max}

can be pretrained, and we are able to fix the invariant part and fine-tune only the private part when switching to a new toolchain. This will result in fine-tuning fewer parameters when transferring to a new toolchain and better efficiency.

4 DS SAMPLER
4.1 Optimization with Cross-Entropy Method

We will now describe the original model-free version of the Cross-Entropy Method (CEM) [15] and how it can be applied to optimize HLS design. It was originally designed for rare-event simulation but can be applied for global optimization, and it has the ability to converge to global optimum theoretically [22, 24]. While the cross-entropy method is treated as a model-based technique in some work [12] because it contains a model for the distribution of rare events (good designs), we treat the original version of the algorithm as model-free since it does not contain a learned model of the reward function. We will show that the cross-entropy method is a strong candidate for efficient global optimization.

Algorithm 1 outlines the application of the Cross-Entropy Method (CEM) for optimizing black-box functions. The process begins by initializing the sampling distribution $p(x)$ to a random distribution (line 3). In each iteration, N samples are drawn from the current distribution and evaluated using the ground truth function R (lines 6, 7). Based on these evaluations, the distribution is then updated to concentrate more on samples that exceed the $(1 - \rho)$ quantile of the N rewards, i.e. focusing on the top $N \cdot \rho$ performing samples (lines 9-11). To maintain randomness and prevent overfitting, the distribution is modified by a step size α, which typically ranges from 0.7 to 0.9. This algorithm is noted for its rapid convergence rate and robustness against different hyperparameters, as introduced in [15, 21].

To tailor Algorithm 1 for HLS design optimization, the key tasks involve modeling the distribution of design configurations and implementing updates to this distribution. In HLS, each design is characterized by a combination of different pragma values. Each pragma within the program must either be assigned a specific value or be set to a default value that effectively disables it, thereby completing the design configuration. We model the distribution over all possible designs by assigning a probability vector $p_i(x)$ for each pragma i, resulting in $p(x) = \Pi_i p_i(x_i)$. The independent
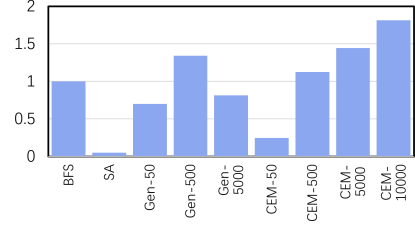

Figure 5: Normalized geomean performance of CEM and other optimization algorithms: The Gen-K and the CEM-K represent the different population size used in the genetic search and CEM
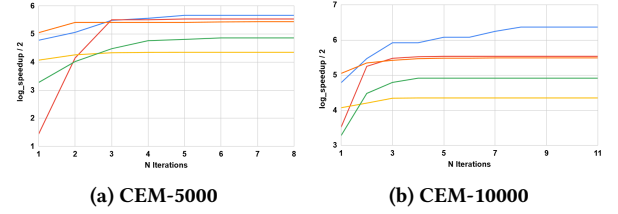

(a) CEM-5000   (b) CEM-10000

Figure 6: The convergence rate of CEM-K. The horizontal axis shows the number of iterations, and the vertical axis depicts the logarithmic speedup. The different lines illustrate different kernels. The performance typically reaches a plateau in less than 10 iterations.

modeling reduces the likelihood of overfitting when the number of data points available for distribution calculation is limited. We will demonstrate that this method of distribution modeling not only simplifies sampling distribution updates but also yields effective optimization results for HLS designs.

To implement the distribution update, we first get the one-hot embedding of each design. Specifically, for each design x, we use a one-hot vector x_i for each pragma, where $x_{ij} = 1$ if the discretized value of pragma i is j, and $x_{ij} = 0$ otherwise. For example, consider a design space containing two pragmas P_a and P_b, and suppose P_a can take value from $[1, 2, 5]$ and P_b can take value from $[1, 2]$. Then the design $x = \{P_a = 5, P_b = 2\}$ will be encoded with $\{x_a = [0, 0, 1], x_b = [0, 1]\}$.

Then, at each iteration, the distribution $\hat{p}$ in Algorithm 1 can be calculated by separately calculating the probability vector p_i of each pragma i, with the one-hot vector x_i:

$$\hat{p}_i = \frac{\sum_{x \in \mathcal{D}_s^t} x_i \cdot \mathbb{1}[R(x) > q]}{\sum_{x \in \mathcal{D}_s^t} \mathbb{1}[R(x) > q]} \tag{1}$$

where R is the ground truth reward function, $\mathcal{D}_s^t$ denotes the N samples generated by the previous distribution p at iteration t, and the q is the $1 - \rho$ quantile of G_r.

When implementing the algorithm, we also memorize the best $\rho \cdot N$ design in all previous iterations and consider them when calculating the quantile and optimizing the distribution, which improves the performance. This detail is omitted in the algorithm description for simplicity.

We first conduct experiments to validate that CEM is good for global optimization, comparing it with other algorithms including breadth-first search (BFS), simulated annealing, and genetic algorithm. Due to the long runtime of evaluating HLS design, we use the

model prediction as the ground truth function. Even though high predicted performance may not result in high actual performance, it is a good indicator of the ability of the optimizer. We evaluate 5 programs with a large design space and calculate the geomean of the top-1 latency under resource constraints. We also test CEM and genetic search with different population sizes. Fig. 5 displays the normalized result with respect to BFS. Gen-K and CEM-K stand for the genetic algorithm and CEM under different population sizes. Under the same runtime budget, the CEM explorer performs the best among all baselines, and the genetic search also achieves a comparable performance. When the population size is larger, the genetic search will run fewer iterations under the same time limit, resulting in the performance degradation from Gen-500 to Gen-5000. On the other hand, CEM could uncover a better design when increasing the population size, due to its fast convergence rate. As illustrated in Fig. 6, the performance typically converges after 8 iterations. However, the result also shows that optimization with CEM is not sample-efficient. It would require a very large population size, typically larger than 1000, to reach a stable performance. When the population size N is small, the distribution may quickly overfit to a specific design. Evaluating more than 1000 designs using HLS is very time-consuming, even if a dozen of designs can be evaluated in parallel. So, we consider inserting model prediction in the loop to improve the sample efficiency of the CEM in the next section.

4.2 Model-based CEM

One simple way to achieve sample efficiency is to replace the evaluation on the true reward function R with a proxy model $\hat{R}_\theta$, pick the design with the top prediction value, and validate it with R at the last iteration. However, such an approach may cause severe performance degradation, because the model $\hat{R}_\theta$ is not accurate in terms of the label shift caused by changing the toolchain, and the distribution shift caused by a sampling distribution update. In Section 1, we illustrate how the model could be vulnerable to the label shift. Another factor to consider is the distribution shift. Even if the model has already adapted to the label shift with data on the new version, it may have low accuracy on some designs visited by the explorer, because of the mismatch between the training data distribution induced by the data gathering heuristic, and the testing distribution induced by the explorer. Therefore, it is possible for the explorer to find a design with a high predicted performance but a low actual performance. Such a phenomenon is constantly observed when trying to solely rely on the proxy model to do the optimization [18, 19]. Intuitively, a powerful optimization algorithm will put pressure on the model to be perfectly accurate on the whole design space, which is usually hard to achieve given limited labeled data.

Therefore, it is critical to first update the model to have a reasonably accurate prediction under the current function $R(x)$ and the current data distribution $p(x)$. Typically, we want to optimize the following objective, where $\mathcal{D}$ is the whole design space.

$$\min_\theta \sum_{x \in \mathcal{D}} p(x)(R(x) - \hat{R}_\theta(x))^2 \qquad (2)$$

When the sampling distribution updates, the objective becomes a weighted error, assigning greater weight to regions that potentially contain high-quality designs. Although it is still sample inefficient

to minimize the error of the model with arbitrary data distribution on the whole design space, we find that it is possible to achieve better sample efficiency by integrating the model update into the optimization process. For the CEM, if the model can have an accurate prediction on $\mathcal{D}_s^t$ in each iteration, then the next distribution can be calculated solely based on the model prediction. Note that $\mathcal{D}_s^t \subset \mathcal{D}$ is sampled from the distribution $p(x)$, and is an unlabeled pool of the design. As discussed in section 4.1, the size of $\mathcal{D}_s^t$ could still be large, so we want to further improve the sample efficiency, by selecting several designs from $\mathcal{D}_s^t$ to update the model. The sub-problem to solve then is: How to select a few designs in $\mathcal{D}_s^t$ and update the model on the true value of these designs, so that the model has a minimal error on $\mathcal{D}_s^t$. More specifically, we want to find $\mathcal{D}_{core}^t \subset \mathcal{D}_s^t$, so that the model can have minimal error on $\mathcal{D}_s^t$ when the parameter θ is updated on $\mathcal{D}_{core}^t$.

4.2.1 Selective Labeling for Model Update. Several heuristics have been proposed to tackle the data selection problem in active learning. Those heuristics primarily consider two objectives, uncertainty and diversity. Both objectives are important to minimize the model error given a fixed labeling budget.

However, the uncertainty-based approaches can only characterize the distribution shift, the change of $p(x)$. In task transfer, it is also important to be aware of the change in the function $R(x)$, or $p(y|x)$. While it is hard to estimate the change without actually observing the true value, it is still reasonable to sample diverse designs under a fixed budget, so that more information can be revealed about the unknown function. Therefore, we choose to use the coreset approach [25] to acquire diverse data.

The coreset based sampling algorithm approximates the model error on the whole pool of data with distance measurement from selected data points and the rest of the data. [25] and [20] separately model the problem as K-Center and K-Means clustering. We consider utilizing the K-Means algorithm on the hidden representation due to its efficiency. Since we have both a regression model $\hat{R}_{\theta_r}$ to predict the QoR and a classification model $\hat{R}_{\theta_c}$ to predict the validity of the design, we need samples to provide information for both models. We thus divide the K labeling budget into K_1 and K_0 and divide the unlabeled pool $\mathcal{D}_s^t$ into two subsets $\mathcal{D}_{s_1}^t$ and $\mathcal{D}_{s_0}^t$. $\mathcal{D}_{s_1}^t$ contains designs that the classification model $\hat{R}_{\theta_c}$ predicts valid, and $\mathcal{D}_{s_0}^t$ contains those that the model predicts invalid. Then, we run K-Means to get K_1 cluster centroids of the hidden embeddings of $\hat{R}_{\theta_r}$ on $\mathcal{D}_{s_1}^t$ and K_0 cluster centroids of the hidden embeddings of $\hat{R}_{\theta_c}$ on $\mathcal{D}_{s_0}^t$. Then, the designs closest to the cluster centroids are selected to be added to the database.

4.2.2 Active-CEM. We now describe the complete exploration algorithm which consists of an outer loop that updates the sampling distribution according to the model prediction, and an inner loop that fits the model to the current distribution. We abbreviate the name with Active-CEM.

As shown in Algorithm 2, the sampling distribution p is initialized to random, and the model $\hat{R}_{\theta_r}$, $\hat{R}_{\theta_c}$ is initialized with the old model trained on the old database. A database B that has designs with true labels is initialized to empty. For each iteration t, an unlabeled pool $\mathcal{D}_s^t$ will be sampled according to distribution p. Then, the unlabeled pool $\mathcal{D}_s^t$, the database B, and the current model will be forwarded to the inner active learning loop with the K-Means

Algorithm 2 Active-CEM

Input: num iterations n, population size N, true function R, quantile ρ, old model $\hat{R}'_{\theta_r}$, $\hat{R}'_{\theta_c}$, active learning round m, batch size b

Output: best design d_{best}, labeled database B

Initialize distribution $p(x)$ to random.

$\hat{R}_{\theta_r}, \hat{R}_{\theta_c} \leftarrow \hat{R}'_{\theta_r}, \hat{R}'_{\theta_c}$

for $t = 1$ **to** n **do**

$\quad \mathcal{D}_s^t \leftarrow N$ samples without replacement from $p(x)$

$\quad$ # actively update model

$\quad$ **for** $j = 1$ **to** m **do**

$\quad\quad \mathcal{D}_{core}^t \leftarrow$ select b samples from $\mathcal{D}_s^t$ $\qquad$ # K-Means

$\quad\quad B = B \cup (\mathcal{D}_{core}^t, R(\mathcal{D}_{core}^t))$ # run HLS on each instance in $\mathcal{D}_{core}^t$, add to B

$\quad\quad \hat{R}_{\theta_r}, \hat{R}_{\theta_c} \leftarrow$ train_model$(B, \hat{R}_{\theta_r}, \hat{R}_{\theta_c})$

$\quad$ **end for**

$\quad$ # use model prediction to update sampling distribution

$\quad$ # invalid points assigned to $-\infty$

$\quad \mathcal{R}_s^t(x) = \hat{R}_{\theta_r}(x) + \infty \cdot (\hat{R}_{\theta_c}(x) - 1)$ for $x \in \mathcal{D}_s^t$

$\quad q \leftarrow$ Calculate the $1 - \rho$ quantile of $\mathcal{R}_s^t$

$\quad$ Update $p(x)$ according to $\mathcal{R}_s^t$ and q

end for

return best design found, B, $(\hat{R}_{\theta_r}, \hat{R}_{\theta_c})$

---

algorithm. The active learning loop will return an updated model. Then, the updated model will predict the label for all designs in $\mathcal{D}_s^t$. According to the predicted label, the $1 - \rho$ quantile of the design performance will be calculated on the valid designs that satisfy the resource constraint, and the distribution will be updated according to Equation 1. The only difference is that the true value $R(x)$ in equation 1 is replaced with the updated prediction model $\hat{R}_{\theta_r}(x)$ and $\hat{R}_{\theta_c}(x)$.

4.2.3 Sample efficiency and runtime of the algorithm. We briefly discuss the sample efficiency and the runtime of the proposed algorithm. By sample efficiency, we mean the number of designs evaluated with the HLS toolchain during exploration, which is the major runtime bottleneck. As shown in algorithm 2, the model-free version of the CEM algorithm needs to sample $N \cdot n$ design points, while the model-based version only requires $m \cdot b \cdot n$. As for the runtime, since the design is evaluated in batch, the runtime is bounded by the number of parallel HLS workers and the number of samples. The overhead of sampling unlabeled design and updating the distribution are negligible. Suppose there are C parallel workers to run HLS, and the timeout for running HLS design is T, then the runtime of the model-free version is $Tn \cdot \left\lceil \frac{N}{C} \right\rceil$, while the runtime of the model-based version is $Tnm \cdot \left\lceil \frac{b}{C} \right\rceil$. While proper scheduling could reduce the overall runtime of evaluating N design with C workers, we provide an upper bound of the runtime here.

4.3 Design Space Pruning

Another way to achieve sample efficiency is to prune away part of the large design space so that the explorer will have less pressure to explore a smaller design space. While design space pruning is not the focus of this work, we empirically find that the "TILE" pragma

Table 2: Program domain and design space size

Program	Domain	DS Size
2mm	multi-MM	4.9E+08
3mm	multi-MM	1.8E+13
atax	MV	2.6E+03
atax-med	MV	9.5E+03
covariance	Data Mining	9.7E+08
fdtd-2d	Stencil	1.8E+10
gemm-p	MM	4.1E+05
gemver	MV & VV	1.0E+11
gemver-med	MV & VV	1.4E+10
jacobi-2d	Stencil	7.6E+06
symm-opt	Symmetric MM	7.3E+05
trmm-opt	Triangular MM	5.0E+04
syr2k	Symmetric rank-2k	3.4E+05

is not as efficient as the "PARALLEL" and "PIPELINE" pragma for optimizing design. Also, whether or not "TILE" pragmas would be effective is determined mostly by the program itself (whether it is memory bound/compute bound), and is more agnostic to the change in the toolchain. Therefore, we consider pruning away "TILE" pragmas for some programs. We adopt a conservative heuristic that prunes away "TILE" pragmas only if (1) less than 10% of the designs in the old database have tiling pragma, (2) the top-10 designs do not contain "TILE" pragma, and (3) the memory footprint of the workload could fit on-chip. We only cut the "TILE" pragma if all these conditions are satisfied. With this heuristic, the model will also witness less distribution shift at the early stage of exploration.

5 EVALUATION

5.1 Experiment setup

We evaluate our approach on two task transfer scenarios: from AMD Xilinx Vitis 2020.2 (V20) to Vitis 2021.1 (V21) and from Vitis 2021.1 (V21) to Vitis 2023.2 (V23). We also utilize the AMD Xilinx Merlin Compiler [10]. We assess our methods on 13 programs from the HLSyn [5] dataset, from toolchain V20 to V21. We do not consider programs that have a very small design space. The workload covers dense linear algebra, data analytics, and stencil operations. To evaluate the robustness of the method to different label shifts, we further test it by transferring from V21 to V23, on 5 programs with very large design space. We list the domain and the size of the design space for each program in Table 2. "MM" stands for matrix-matrix multiplication, and "MV" stands for matrix-vector multiplication. Since we focus on assessing task transfer to new toolchains, we fixed the evaluation platform to Xilinx U200 with a frequency of 250MHz.

While the proposed model architecture introduces extra overhead, we simply finetune the HARP model during the CEM exploration. After collecting a database with the explorer, we train the new model architecture with invariant embedding on the database and conduct one last round of DSE with a BFS search. The model is trained with a batch size of 64 and is trained for 1500 epochs, and the learning rate is set to 1e-3 with a linear warmup and is cosine annealed to 1e-5. When training the model with invariant embedding, we randomly sample mini-batches of 64 from the V18 and the V20 database, and forward all three batches in a single

Table 3: V21 transfer learning model accuracy

Model	Validation RMSE	Test RMSE
HARP Scratch	0.361 (0.042)	0.493 (0.031)
HARP Finetune	0.332 (0.014)	0.481 (0.013)
Inv-Hybrid	**0.285 (0.031)**	**0.432 (0.006)**

iteration to calculate the loss. We run each experiment three times with different random seeds, under the same split of the dataset, and report the mean and standard deviation of each method.

5.2 Model accuracy

We validate the effectiveness of learning invariant embedding on the HLS designs with model accuracy. We compare the proposed model (Inv-Hybrid) with three baselines, the HARP model trained from scratch (HARP scratch) on the new version data, the HARP model initialized with pre-trained parameters on old data and fine-tuned on new data (HARP finetune). For the old version data, we utilize the HLSyn database, which consists of data collected by AutoDSE on V18 and V20 of the toolchain. For the new version data, we utilize the data collected by AutoDSE on the V21 toolchain for the 13 programs we selected, following the data collection pipeline as described in HLSyn. 3975 regression data is gathered for the 13 programs on the V21 toolchain. And the V18 and V20 databases contain 9439 and 5194 data. We split the V21 data into training, validation, and test set, with a portion of 40%, 10%, and 50%, to mimic the scenario where a few data is gathered on the large design space. We select the model with a minimum validation error and record the testing accuracy.

Table 3 summarizes the efficiency of each model, using the regression RMSE as a metric. We observe that the HARP model performs much better when initialized with pre-trained parameters, which is consistent with the observation made previously in [27]. With the Inv-Hybrid model, the RMSE on the test set further improves by 11% and shows better stability during training with less standard deviation. This validates our assumption that we should model both the invariant and the difference between toolchains to achieve good transfer learning accuracy.

5.3 DSE Performance

We next verify the efficiency of the Active-CEM algorithm. We compare with AutoDSE and HARP. For the AutoDSE baseline, we follow previous work and run all three explorers in parallel. Following [27], we set a timeout of 24 hours and do not terminate the jobs that are already running, resulting in a maximum runtime of 25 hours. For baseline evaluation, we adopt the same approach as HARP for running its explorer. Specifically, after collecting data with AutoDSE, we initialize the model with the pre-trained parameter on the old version, and then fine-tune the HARP model on the new version data. Then we run a BFS search for 1 hour with HARP and validate the design with top-10 predicted performance using the HLS toolchain. When transferring to V21, we initialize the model with parameters pre-trained on the V20 database. When transferring to V23, we initialize the model with parameters pre-trained on the V21 database. For a fair comparison, when pretraining the model on existing versions of databases, we only utilize the database collected by AutoDSE. For Active-CEM, similar to HARP, we initialize the model pre-trained parameter on the old version data. For each program, we run Active-CEM for 8 iterations to converge,

Table 4: V21 DSE Speedup & Sample Efficency. We use abbreviation of the baselines. A: AutoDSE; A+H: AutoDSE+HARP; Ac-C: Active-CEM; Ac-C+H: Active-CEM+HARP; Ac-C+Inv: Active-CEM+Inv-Hybrid

Kernel	A	A+H	Ac-C	Ac-C+H	Ac-C+Inv
average	1.00	1.98	2.35	2.38	2.37
geomean	1.00	1.26	1.44	1.48	1.46
sample valid	3975	-	2215	-	-
sample total	17743	-	3081	-	-

Table 5: V23 DSE Speedup & Sample Efficiency

	A	A+H	A-10h	Ac-C
average	1.00	1.02	0.93	1.30
geomean	1.00	1.02	0.93	1.27
sample valid	2786	-	1356	803
sample total	10524	-	3904	1189

and query 30 labeled designs at each iteration. We gather all the labeled data queried so far and finetune the model for 30 epochs in each iteration. The overall runtime of the algorithm is in between $8.9 - 9.1$ hours. It is composed of a $8 \cdot 65$ minutes overall timeout for running HLS and the overhead of the policy and the model update. The runtime is $2.7x$ faster than AutoDSE. The variation in runtime is caused by generating graph data of different sizes and fine-tuning the model.

Table 4 and Table 5 illustrate the result of the design performance when transferring from V20 to V21 and from V21 to V23. When transferring to V21, exploration with Active-CEM performs 2.34× better than AutoDSE and 1.18× better than the best of AutoDSE and HARP and achieves this with a 5.75× total sample efficiency and a 1.74× sample efficiency in terms of valid data. Note that Active-CEM tends to explore more valid designs during policy updates while AutoDSE does not contain heuristics to explicitly sample valid designs, therefore a larger portion of the designs explored by AutoDSE are invalid. As for design space pruning, among all 13 programs, only the design spaces of 3 programs (gemver, 3mm, and fdtd-2d) are pruned without searching for "tile" pragmas. We leave learning a more robust design space pruning strategy to future work. When transferring to V23, Active-CEM still performs 1.3× better than AutoDSE, and 1.27× better than the best of AutoDSE and HARP, and is a 8.85× sample efficiency in total and a 3.47× sample efficiency in terms of valid design, demonstrating the robustness of the proposed method. Better design performance and better sample efficiency are achieved by inserting model prediction into an importance sampling method, utilizing the knowledge learned on the old toolchains and addressing the label and distribution shift.

We further train the HARP model and the Invariant-hybrid model with the database collected by the CEM method, and conduct one last round of DSE with a BFS search. Our result in Table 4 reveals that we can locate a better design in one last round of the exploration, achieving another moderate $1.01x$ speedup on average.

We further perform an ablation study by running AutoDSE for the same amount of time as Active-CEM. Specifically, we pick the checkpoint of AutoDSE at 10 hours and report the best design found by all three explorers. As seen in Table 5, Active-CEM outperforms AutoDSE on average by 1.39× when the runtime budget is the same.

Zijian Ding, Atefeh Sohrabizadeh, Weikai Li, Zongyue Qin, Yizhou Sun, Jason Cong

Table 6: Model accuracy by freezing parameters

Model	Validation RMSE	Test RMSE
HARP FT No Frz	0.332 (0.014)	**0.481 (0.013)**
HARP FT Frz(1)	0.401 (0.07)	0.512 (0.039)
HARP FT Frz(2)	0.367 (0.026)	0.501 (0.024)
HARP FT Frz(3)	0.347 (0.028)	0.509 (0.011)
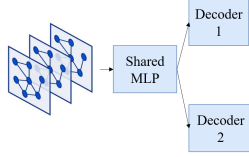
Figure 7: Inv-share: Model architecture for ablation study

Table 7: Ablation study on invariance modeling

Model	Validation RMSE	Test RMSE
Inv-Share	0.358 (0.065)	0.481 (0.044)
Inv-Hybrid	0.285 (0.031)	0.432 (0.006)

Table 8: Ablation study on pruning, toolchain V21

Kernel	Pruned	Not Pruned	DS Size Reduction
gemver	1.00	0.37	15×
fdtd-2d	1.00	0.92	39×
3mm	1.00	0.88	244×

5.4 Ablation Studies

5.4.1 Model transfer by freezing parameters. Table 6 shows the result of transferring the model by fixing the first several layers of the GNN encoder. Similar to the previous experiment, we split the data with 40%, 10%, and 50%, conduct each experiment 3 times with different random seeds, and report the mean and standard deviation. We observe that the model achieves the best accuracy when we do not freeze any parameter, which is different from the conclusion drawn from various transfer learning works with image data[20]. We hypothesize that this is because of the large label shift caused by switching to a different toolchain, and the lack of proper pretraining on a large database that contains multiple versions of data.

5.4.2 Separate modeling the invariance and the difference. We add another baseline to confirm the effectiveness of the proposed model architecture, which models the invariance and the difference between toolchains. Specifically, we implement a model architecture as shown in Fig. 7, which simply uses a different decoder to decode a shared embedding to different outputs. Moreover, we fix the parameter size of both architectures to be the same, so that we can strictly study the effect of decoupling the shared and the private embedding. In Table 7, the test RMSE of "Inv-Hybrid" is lower than "Inv-Share" by 11%, demonstrating the importance of modeling both the invariance and the difference between toolchains.

5.4.3 Design space pruning. We evaluate the efficiency of design space pruning by comparing the design performance when enabling and disabling it. Only 3 out of 13 programs satisfy the pruning constraints and we report the performance difference for them. Table 8 illustrates how the design performance improves when design space pruning is enabled.

Table 9: Domain transfer

(a) Test split 1

Kernel	A+H	Ac-C	Ac-C+H
gemver-med	1.00	1.73	1.73
fdtd-2d	1.00	1.27	1.27
syr2k	1.00	0.94	0.94
trmm-opt	1.00	1.45	1.45

(b) Test split 2

Kernel	A+H	Ac-C	Ac-C+H
3mm	1.00	0.87	0.99
mvt	1.00	0.99	0.99
covariance	1.00	0.72	0.72
gemm-p	1.00	1.04	1.07

5.5 Extending to Domain Transfer

We further test the efficiency of the Active-CEM algorithm by applying it in the domain transfer learning setting. Specifically, we split the HLSyn dataset, train the HARP model on the training kernels, and test it on 4 testing kernels covering stencil and linear algebra workloads. We evaluate on two different train/test split, on 8 test programs in total. While HLSyn contains variants of the same program, we ensure that the training set and the test set contain completely different programs. Table 9 demonstrates that on test split 1, Active-CEM outperformed the best of AutoDSE and HARP on 3 out of 4 kernels, achieving a 2.7× runtime speedup. Similarly, on test split 2, it demonstrated equal or superior performance on 3 out of 4 kernels. The performance variance across different test kernels can be attributed to the size of the design space and its difference from the training set. For instance, the training set included only one data-mining kernel, resulting in poorer performance on the covariance kernel compared to other linear algebra kernels.

6 CONCLUSION

In this work, we discuss the sample efficiency challenge in the task transfer learning problem for HLS DSE. We mitigate the challenge with a novel approach on both sampling and modeling. On sampling, we propose Active-CEM, a model-based explorer that efficiently samples high-quality designs in the whole design space. On modeling, we propose to model both the invariance and difference between toolchains and jointly train a single model with data collected from multiple toolchains. Moving forward, we plan to evaluate our sampling method in the domain-transfer setting and consider task-transfer learning for different devices. We also plan to combine domain knowledge in the sampling process for better efficiency. In [16], we extend our task transfer results to the complete HLSyn dataset on V21 toolchain.

ACKNOWLEDGMENTS

REFERENCES

[1] AMD/Xilinx. 2020. Vitis HLS 2020.2. https://docs.amd.com/r/2020.2-English/ug1399-vitis-hls/Introduction-to-Vitis-HLS.

[2] AMD/Xilinx. 2021. Vitis HLS 2021.1. https://docs.amd.com/r/2021.1-English/ug1399-vitis-hls/Getting-Started-with-Vitis-HLS.

[3] AMD/Xilinx. 2023. Vitis HLS 2023.2. https://docs.amd.com/r/en-US/ug1399-vitis-hls/Introduction.

[4] Tianyi Bai, Yang Li, Yu Shen, Xinyi Zhang, Wentao Zhang, and Bin Cui. 2023. Transfer learning for Bayesian optimization: A survey. *arXiv preprint arXiv:2302.05927* (2023).

[5] 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. In *Thirty-seventh Conference on Neural Information Processing Systems Datasets and Benchmarks Track*.

[6] Yunsheng Bai, Atefeh Sohrabizadeh, Yizhou Sun, and Jason Cong. 2022. Improving GNN-based accelerator design automation with meta learning. In *Proceedings of the 59th ACM/IEEE Design Automation Conference*. 1347–1350.

[7] Ahmet F Budak, Keren Zhu, and David Z Pan. 2023. Practical Layout-Aware Analog/Mixed-Signal Design Automation with Bayesian Neural Networks. In *2023 IEEE/ACM International Conference on Computer Aided Design (ICCAD)*. IEEE, 1–8.

[8] Yuze Chi, Weikang Qiao, Atefeh Sohrabizadeh, Jie Wang, and Jason Cong. 2022. Democratizing domain-specific computing. *Commun. ACM* 66, 1 (2022), 74–85.

[9] Jason Cong, Zhenman Fang, Muhuan Huang, Peng Wei, Di Wu, and Cody Hao Yu. 2019. Customizable Computing—From Single Chip to Datacenters. *Proc. IEEE* 107, 1 (2019).

[10] Jason Cong, Muhuan Huang, Peichen Pan, Yuxin Wang, and Peng Zhang. 2016. Source-to-source optimization for HLS. *FPGAs for Software Programmers* (2016), 137–163.

[11] 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.

[12] Andre Costa, Owen Dafydd Jones, and Dirk Kroese. 2007. Convergence properties of the Cross-Entropy Method for discrete optimization. *Operations Research Letters* 35, 5 (2007), 573–580.

[13] Quanyu Dai, Xiao Shen, Xiao-Ming Wu, and Dan Wang. 2019. Network Transfer Learning via Adversarial Domain Adaptation with Graph Convolution. *CoRR* abs/1909.01541 (2019). arXiv:1909.01541 http://arxiv.org/abs/1909.01541

[14] William J Dally, Yatish Turakhia, and Song Han. 2020. Domain-specific hardware accelerators. *Commun. ACM* 63, 7 (2020), 48–57.

[15] Pieter-Tjerk De Boer, Dirk P Kroese, Shie Mannor, and Reuven Y Rubinstein. 2005. A tutorial on the cross-entropy method. *Annals of operations research* 134 (2005), 19–67.

[16] Zijian Ding, Atefeh Sohrabizadeh, Weikai Li, Zongyue Qin, Yizhou Sun, and Jason Cong. 2024. Efficient Task Transfer for HLS DSE. arXiv:2408.13270 [cs.AR] https://arxiv.org/abs/2408.13270

[17] Yaroslav Ganin, Evgeniya Ustinova, Hana Ajakan, Pascal Germain, Hugo Larochelle, François Laviolette, Mario March, and Victor Lempitsky. 2016. Domain-adversarial training of neural networks. *Journal of machine learning research* 17, 59 (2016), 1–35.

[18] Leo Gao, John Schulman, and Jacob Hilton. 2023. Scaling laws for reward model overoptimization. In *International Conference on Machine Learning*. PMLR, 10835–10866.

[19] Michael Janner, Justin Fu, Marvin Zhang, and Sergey Levine. 2019. When to trust your model: Model-based policy optimization. *Advances in Neural Information Processing Systems* 32 (2019).

[20] Yibo Lin, Meng Li, Yuki Watanabe, Taiki Kimura, Tetsuaki Matsunawa, Shigeki Nojima, and David Z Pan. 2018. Data efficient lithography modeling with transfer learning and active data selection. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 38, 10 (2018), 1900–1913.

[21] Shie Mannor, Reuven Y Rubinstein, and Yohai Gat. 2003. The Cross Entropy Method for fast policy search. In *Proceedings of the 20th International Conference on Machine Learning (ICML-03)*. 512–519.

[22] Leonid Margolin. 2005. On the convergence of the cross-entropy method. *Annals of Operations Research* 134 (2005), 201–214.

[23] Valerio Perrone, Huibin Shen, Matthias W Seeger, Cedric Archambeau, and Rodolphe Jenatton. 2019. Learning search spaces for bayesian optimization: Another view of hyperparameter transfer learning. *Advances in Neural Information Processing Systems* 32 (2019).

[24] Reuven Y Rubinstein and Dirk P Kroese. 2004. *The cross-entropy method: a unified approach to combinatorial optimization, Monte-Carlo simulation, and machine learning*. Vol. 133. Springer.

[25] Ozan Sener and Silvio Savarese. 2017. Active learning for convolutional neural networks: A core-set approach. *arXiv preprint arXiv:1708.00489* (2017).

[26] 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.

[27] 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.

[28] 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.

[29] Qi Sun, Tinghuan Chen, Siting Liu, Jianli Chen, Hao Yu, and Bei Yu. 2022. Correlated multi-objective multi-fidelity optimization for HLS directives design. *ACM Transactions on Design Automation of Electronic Systems (TODAES)* 27, 4 (2022), 1–27.

[30] Ecenur Ustun, Chenhui Deng, Debjit Pal, Zhijing Li, and Zhiru Zhang. 2020. Accurate operation delay prediction for FPGA HLS using graph neural networks. In *Proceedings of the 39th international conference on computer-aided design*. 1–9.

[31] 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.

[32] 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.

[33] Qitian Wu, Hengrui Zhang, Junchi Yan, and David Wipf. 2022. Handling Distribution Shifts on Graphs: An Invariance Perspective. In *International Conference on Learning Representations (ICLR)*.

[34] Shirley Wu, Kaidi Cao, Bruno Ribeiro, James Zou, and Jure Leskovec. 2024. Graph-METRO: Mitigating Complex Graph Distribution Shifts via Mixture of Aligned Experts. arXiv:2312.04693 [cs.LG] https://arxiv.org/abs/2312.04693

[35] Yizhou Zhang, Guojie Song, Lun Du, Shuwen Yang, and Yilun Jin. 2019. DANE: Domain Adaptive Network Embedding. In *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI-19*. International Joint Conferences on Artificial Intelligence Organization, 4362–4368. https://doi.org/10.24963/ijcai.2019/606

[36] Qi Zhu, Carl Yang, Yidan Xu, Haonan Wang, Chao Zhang, and Jiawei Han. 2020. Transfer learning of graph neural networks with ego-graph information maximization. *arXiv preprint arXiv:2009.05204* (2020).