Hierarchical Mixture of Experts: Generalizable Learning for High-Level Synthesis

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

University of California, Los Angeles
weikaili@cs.ucla.edu, allenwang2333@gmail.com, {bradyd, atefehsz, qinzongyue, cong, yzsun}@cs.ucla.edu

Abstract

High-level synthesis (HLS) is a widely used tool in designing Field Programmable Gate Array (FPGA). HLS enables FPGA design with software programming languages by compiling the source code into an FPGA circuit. The source code includes a program (called "kernel") and several pragmas that instruct hardware synthesis, such as parallelization, pipeline, etc. While it is relatively easy for software developers to design the program, it heavily relies on hardware knowledge to design the pragmas, posing a big challenge for software developers. Recently, different machine learning algorithms, such as GNNs, have been proposed to automate the pragma design via performance prediction. However, when applying the trained model on new kernels, the significant domain shift often leads to unsatisfactory performance. We propose a more domain-generalizable model structure: a two-level hierarchical Mixture of Experts (MoE), that can be flexibly adapted to any GNN model. Different expert networks can learn to deal with different regions in the representation space, and they can utilize similar patterns between the old kernels and new kernels. In the low-level MoE, we apply MoE on three natural granularities of a program: node, basic block, and graph. The high-level MoE learns to aggregate the three granularities for the final decision. To train the hierarchical MoE stably, we further propose a two-stage training method to avoid expert polarization. Extensive experiments verify the effectiveness of the proposed hierarchical MoE. We publicized our codes at https://github.com/weikai-li/HierarchicalMoE.

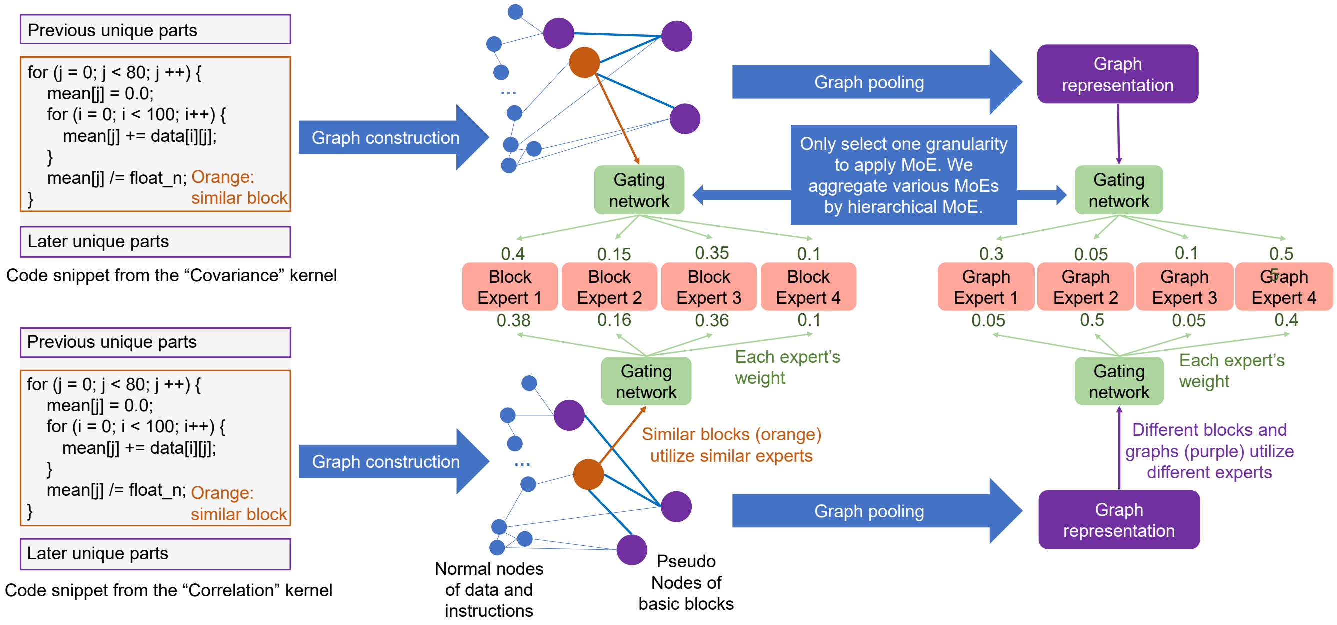
Introduction

With the heated demand for domain-specific accelerators, field-programmable gate arrays (FPGA) are widely used. It is, however, labor-intensive to write register-transfer-level hardware description languages, such as VHDL and Verilog. High-level synthesis (HLS) provides a much easier solution by compiling a source code written in a software programming language, such as C and MatLab, into an FPGA circuit (Cong et al. 2011, 2022; Schafer and Wang 2019).

The source code consists of a program (also called "kernel") that describes the FPGA's functional behaviors, and several pragmas inserted in the program that instruct the hardware synthesis process, such as parallelization, pipeline, etc., as illustrated in the code snippets in Fig. 1. While it is relatively easy to design the program, it heavily requires hardware knowledge to design the pragmas, and different pragma designs can lead to significantly different FPGA performances, posing a great challenge for software developers (Sohrabizadeh et al. 2022). Recent works have automated pragma design for programs, employing heuristics methods or machine learning methods. The heuristic methods use bottleneck analysis (Sohrabizadeh et al. 2021) or non-linear programming based on lower-bound objective function (Pouget, Pouchet, and Cong 2024a,b). The machine learning methods train a surrogate model to predict the FPGA's performance from the source code, so that we can rely on the model prediction to find the best pragmas without running time-consuming HLS. Since HLS data is very scarce, it is difficult to train a large language model, and graph neural networks (GNNs) that are based on the control data flow graph (Sohrabizadeh et al. 2022; Bai et al. 2022; Sohrabizadeh et al. 2023; Ustun et al. 2020; Wu, Xie, and Hao 2023; Wu et al. 2022a) are widely utilized. Some very recent work (Qin et al. 2024) explores using both GNN and lightweight language models.

While machine learning models have better learning ability than the heuristic methods, they do not generalize well to unseen kernels. In real-world applications, we often encounter new circuit design requirements. The model trained on existing kernels usually fails to perform well on new kernels. This can be viewed as a domain generalization (Bai et al. 2022; Kwon and Carloni 2020) problem, where each kernel is a domain. Note it is very time-consuming to run HLS to acquire the labels on new kernels, with each run taking minutes to hours (Sohrabizadeh et al. 2023), thus data-efficient fine-tuning is required. Domain generalization methods for GNN usually employ adversarial training to align the representation space between different domains (Dai et al. 2019; Zhang et al. 2019; Wu et al. 2020; Shen et al. 2023), or do data augmentation to learn invariant representation under risk extrapolation (Wu et al. 2022b; Liu et al. 2023a). However, these approaches are not applicable to the HLS prediction task, as the differences between kernels are informative for the prediction, so forcing them to the same distribution does not work.

Nonetheless, a unique opportunity of the HLS prediction task is that kernels usually share some similar substructures, as shown in Fig. 1. It could be beneficial if we can leverage



```
for (j = 0; j < 80; j ++) {
    mean[j] = 0.0;
    for (i = 0; i < 100; i++) {
        mean[j] += data[i][j];
    }
    mean[j] /= float_n;
}
```

Figure 1: Motivation of utilizing MoE. The two kernels share a similar nested loop, while the other parts are different. Similar parts can utilize similar experts via similar gating, while different parts can utilize different experts. This is just an illustration. Actually, we do not apply MoE on different granularities in the same model in this way. We aggregate them like Figure 2.

such similarities. We thus propose a two-level hierarchical mixture of experts (MoE) to address this issue. MoE (Jacobs et al. 1991; Shazeer et al. 2017) uses several expert networks instead of a single fixed network, and it computes the weighted sum of their outputs. Different experts can specialize in different regions in the hidden representation space, so that similar substructures in different domains can be processed by the same experts. A gating network computes the expert assignment, which learns which expert should deal with which substructure. GraphMETRO (Wu et al. 2024) applies MoE to improve GNNs’ generalizability by predefining a set of domain shifts and training each expert to deal with one type of domain shift by data augmentation. We take an orthogonal direction, eliminate the need for hardware knowledge to define the domain shifts, and focus on the model structure rather than data augmentation.

Another unique opportunity for HLS prediction is that the input graph naturally has three levels of granularity: nodes, blocks, and the whole graph. MoE can be constructed on either of these three granularities. A natural question is which granularity to use. It is a hard task as different kernels benefit from different granularities. Simply applying MoE on all three granularities in one model does not work. Therefore, we propose a higher-level MoE to aggregate the three low-level MoEs according to the need. Nonetheless, optimization with the hierarchical MoE is challenging. Since the three low-level MoEs are different, training easily leads to expert polarization. We propose a two-stage training to stabilize the training. To the best of our knowledge, while some papers (Shazeer et al. 2017) have studied hierarchical MoE to improve memory efficiency, this is the first study that demonstrates the performance gain of hierarchical MoE. We conduct extensive experiments on the largest HLS benchmark dataset, HLSyn (Bai et al. 2023), and the experiment results reveal the effectiveness of hierarchical MoE. In summary, we make the following contributions:

- We identify the difficulties of domain generalizable learning in HLS prediction and formalize this problem.
- We propose hierarchical MoE to address these challenges and a two-stage training approach to stabilize its training.
- Extensive experiments verify the effectiveness of our methods, where the average speedup of the FPGA design is 26.6% higher than that of the baseline method.

Related Work

Machine Learning for HLS Prediction

There are two directions for HLS pragma design automation. The first direction is heuristic-driven. AutoDSE (Sohrabizadeh et al. 2021) uses bottleneck-guided searching based on the HLS feedback, and it is well-generalizable. Nonlinear programming performs well on affine kernels based on a lower-bound objective function (Pouget, Pouchet, and Cong 2024a,b). The second direction is data-driven, which trains a surrogate machine learning model to predict the FPGA’s latency and resource utilization. GNNs (Sohrabizadeh et al. 2022, 2023; Ustun et al. 2020; Wu et al. 2022a; Wu, Xie, and Hao 2023, 2021; Qin et al. 2024) are widely used. While machine learning models could perform better than heuristic methods with sufficient training data, they suffer from poor generalizability. Several previous works have tried to solve this issue. (Ferretti et al. 2020) transforms the best pragma design from the most similar source kernel to the target kernel, but its transformation is too simple and highly limited. (Kwon and Carloni 2020) trains a separate domain adaptor for each domain, while (Bai et al. 2022) uses the meta-learning method, MAML (Finn, Abbeel, and Levine 2017), to find a generalizable parameter initialization. However, since we have many kernels, it is difficult to train MAML to get stable results. We only have hundreds of data for each kernel, so it is difficult to train an adaptor. Instead, we explore an orthogonal direction: to improve the model structure's generalizability.

Domain Generalization for Graph Neural Network

Domain generalization aims to reduce the performance gap between the source and target domains. Most related works (Dai et al. 2019; Zhang et al. 2019; Wu et al. 2020; Shen et al. 2023) employ adversarial training to align the representation space between the source and target domains. Some papers (Wu et al. 2022b; Liu et al. 2023a) design specific data augmentation methods and learn invariant representations under risk extrapolation. However, we cannot directly align the representation space of different kernels since their difference is large and is useful for the prediction. GraphMETRO (Wu et al. 2024) predefines several types of domain shifts and uses MoE to deal with them, where each expert deals with one type. We take an orthogonal direction and eliminate the need for hardware knowledge to define the types of domain shifts, and our model based on the three granularities is specifically designed for this task.

Mixture of Experts (MoE)

MoE uses several expert networks and calculates the weighted sum of their outputs (Jacobs et al. 1991; Shazeer et al. 2017). It is useful in domain transfer learning in computer vision (Li et al. 2023; Zhong et al. 2023). It has also been utilized in GNN to improve the performance (Wang et al. 2023; Han et al. 2024), diversify node representations in fairness-augmented graphs (Liu et al. 2023b), and deal with class imbalance (Hu et al. 2021). Previous work (Shazeer et al. 2017) has employed hierarchical expert routing to improve memory efficiency, enabling a larger number of experts. However, previous papers have not shown the performance gain of hierarchical MoEs over regular MoEs when using the same number of total experts.

Preliminary

Task Definition

HLS prediction is formalized as a graph regression task. Following previous works (Sohrabizadeh et al. 2022, 2023), we utilize ProGraML (Cummins et al. 2021) graph to represent a source code, which is a control data flow graph. Nodes represent instructions, variables, and constant values, and edges represent the control flow and data flow. A GNN model is trained to predict the FPGA's latency and the utilization of four resources: LUT, FF, DSP, and BRAM. In the domain generalization setting, we train the model on N source kernels $D^{(train)} = \{D_1, D_2, ..., D_N\}$, where $D_i = \{(X_i, T_{i1}, Y_{i1}), (X_i, T_{i2}, Y_{i2}), ..., (X_i, T_{ii_n}, Y_{ii_n})\}$ is the i-th kernel containing i_n samples, and each sample consists of a program X, a pragma design T, and a label Y. For a new kernel D_{test}, we consider the constraint of data scarcity where we only use k samples from the dataset $D^{(k)} \subset D_{test}, |D^{(k)}| = k$, to fine-tune the model. After fine-tuning, there are two ways of evaluation: offline evaluation and online evaluation. In the offline evaluation, we evaluate the MSE on the left-out test samples: $D_{test} \setminus D^{(k)}$. In the online evaluation, we do design space exploration (DSE) to search for good pragma designs. Based on the fine-tuned model's prediction, we select the top M designs to run HLS, and evaluate the speedup (M is set to 10 in our experiments).

HARP Model

The proposed hierarchical MoE can be adapted to any GNN model. We use one of the SOTA GNN models for HLS prediction, HARP (Sohrabizadeh et al. 2023), as the base model. Since the original ProGraML graph is not good at modeling long-range dependency, HARP creates a "pseudo node" for each basic block and connects it with every node within that basic block. A basic block is a sequence of instructions with a single entry point and a single exit point where the terminator instruction can be a branch, return, etc. We illustrate HARP's model structure in the appendix. We use V to denote the set of all nodes and V_B to denote the set of pseudo nodes. HARP consists of five components: (1) GNN encoder: it consists of 6 GNN layers and learns node representation; (2) Pragma MLP: it uses an MLP for each pragma type to transform the representation of pseudo nodes that are modified by that pragma type; (3) Another GNN layer after pragma MLP: it further updates the node representations; (4) Graph pooling: it performs graph pooling on pseudo nodes V_B to form one graph representation and on normal nodes $V \setminus V_B$ to form another, then concatenates the two; (5) Output MLP: it makes the final prediction.

Methods

Low-Level Mixture of Experts

A unique opportunity for the HLS prediction task is that the input graph has three granularities: normal nodes that represent data and instructions, pseudo nodes that represent basic blocks, and the whole graph that represents a source code file. If a data point from the target kernel shares a similar statement (node), basic block, or the whole code (graph) with a data point from the training kernels, MoE could be helpful. Thus, we consider employing MoE on the three granularities: MoE on all nodes including normal nodes and pseudo node (node MoE), MoE only on pseudo nodes (block MoE), and MoE on the graph (graph MoE). Based on our pre-exploration, the best practice is to apply MoE in the components after the pragma MLP, since the experts need to share the same encoder and pragma MLPs. In our pre-exploration, we found that the best structure of the gating network is a linear layer. The model structures are shown in Figure 2.

Node MoE. For the MoE that operates on all nodes, we apply it on the GNN layer after the pragma MLP. We train n GNN layers of the same structure as n experts. We denote the representation of node v_i after the pragma MLP as $\boldsymbol{h_i} \in \mathbb{R}^d$. The node MoE is formulated as:

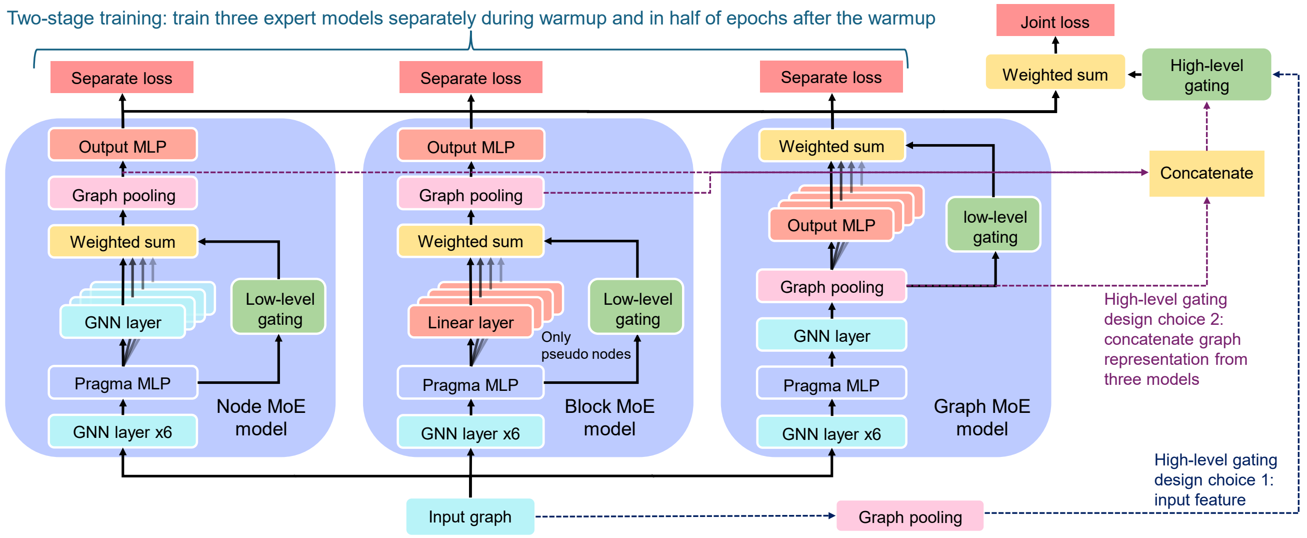


Figure 2: Illustration of hierarchical MoE. The three low-level models are aggregated via the high-level gating network.

$$\boldsymbol{h_i'} = \sum_{j=1}^{n} softmax(\boldsymbol{W_G^{node}} \boldsymbol{h_i})_j \cdot GNN_j(\boldsymbol{h_i}, \{h_j, j \in N(v_i)\}), \tag{1}$$

where $\boldsymbol{W_G^{node}} \in \mathbb{R}^{n \times d}$ is the parameter of the gating network, $N(v_i)$ is the neighbors of v_i, and $GNN_j(\cdot)$ is the j-th expert. The $softmax$ function is a commonly-used normalization method that ensures the weights of all experts sum up to 1. The expert weights can be seen as "attention". We calculate the expert weights (attention scores) using the gating network based on the node embeddings $\boldsymbol{h_i}$. By employing the MoE layer, different nodes can utilize different parameters for message passing and message aggregation.

Block MoE. We have tried applying MoE on the pragma MLP, but the performance has decreased. It might be because each pragma type has a separate pragma MLP, so the training data for each pragma MLP is already limited. If we apply MoE, it will dilute the training data even more. Instead, we add an additional MoE layer on the pseudo nodes' representations after the pragma MLP, so that we can increase the expressiveness of the pseudo nodes' representations before graph pooling. The representation space of the pseudo nodes after the block MoE could be different from normal nodes, so it is no longer suitable to have another GNN layer after the pragma MLP to do message-passing between pseudo nodes and normal nodes. We have also tried applying an additional layer for normal nodes after the pragma MLP, but the performance is unsatisfactory. It might be because the normal nodes lack a global view, so it is not useful to further transform its representation. After having the block MoE, performing graph pooling only on the pseudo nodes performs better than graph pooling on both pseudo nodes and normal nodes. Therefore, we do not utilize normal nodes after the GNN encoder in the block MoE model. Ablation study of various designs is in the appendix.

We use n linear layers as n expert networks. We denote the representation of pseudo node v_i after the pragma MLP as $\boldsymbol{h_i} \in \mathbb{R}^d$. The block MoE is formulated as:

$$\boldsymbol{h_i'} = \sum_{j=1}^{n} softmax(\boldsymbol{W_G^{block}} \boldsymbol{h_i})_j \cdot \boldsymbol{W_j} \boldsymbol{h_i} \ \ (v_i \in V_B), \tag{2}$$

where $\boldsymbol{W_G^{block}} \in \mathbb{R}^{n \times d}$ is the parameter of the gating network, and $\boldsymbol{W_j} \in \mathbb{R}^{d \times d}$ is the j-th expert network.

Graph MoE. We apply the graph MoE on the output MLP. We denote the graph representation after graph pooling as $\boldsymbol{h_G}$. Different from the block MoE structure, here we use the original pragma MLP and graph pooling structures as HARP. We employ the output MLP as the expert network. The graph MoE is formulated as:

$$\hat{Y}^{(t)} = \sum_{j=1}^{n} softmax(\boldsymbol{W_G^{graph}} \boldsymbol{h_G})_j \cdot MLP_j^{(t)}(\boldsymbol{h_G}), \tag{3}$$

where $W_G^{graph} \in \mathbb{R}^{n \times d}$ is the parameter of the gating network, $MLP_j^{(t)}(\cdot)$ is the j-th expert for predicting the objective t (t could be latency or a certain resource's utilization). The expert assignment is the same for all prediction objectives. The MLP contains four linear layers and the ELU activation function in between. Different design points can utilize different experts to make the final prediction.

Regularization term. As discovered in many literature (Shazeer et al. 2017; Wang et al. 2023; Li et al. 2023), expert polarization is a common problem of MoE. Due to the random initialization of expert networks, different experts initially perform differently. The gating network learns to assign higher weights to better experts. It results in more training of the initial better experts, leading to their even better performance, which forms a cycle. As training continues, the MoE model might collapse and only use the best expert. To avoid this issue, we apply a regularization term commonly used in MoE (Shazeer et al. 2017):

$$L_R = CV(\boldsymbol{I(W_G)}),\ \boldsymbol{I(W_G)} = \sum_{i=1}^{M} softmax(\boldsymbol{W_G h_i}), \quad (4)$$

where $CV(\cdot)$ is the coefficient of variation. We calculate the importance score $\boldsymbol{I(W_G)} \in \mathbb{R}^n$ for gating network W_G as the total weights assigned to each expert. For node/block MoE, M is the total number of nodes/pseudo nodes in all graphs; for graph MoE, M is the number of graphs. This regularization encourages balanced expert assignments.

High-Level Mixture of Experts

Now we have three MoE models operating on different granularities. A simple approach to combining them is to use all of them together in one model. However, our ablation study verifies that the performance will be worse. This demonstrates that we only need MoE on one granularity in one model. However, the best granularity greatly varies for different kernels, and it is difficult to discover a pattern. Therefore, we propose a high-level MoE to aggregate them. It calculates the weighted sum of the outputs of the three low-level MoE models.

We propose two designs of the high-level gating network, as illustrated in Figure 2. The first design is to perform graph pooling on the input node features to form a graph representation as the input to the high-level gating network. We use the self-attention graph pooling. Denoting the input feature of node v_i as $\boldsymbol{x_i}$, the graph pooling is formalized as:

$$\boldsymbol{x_G} = \sum_{i \in V} softmax(MLP(\boldsymbol{x_i})) \cdot \boldsymbol{x_i} \quad (5)$$

where X_G is the aggregated input feature, and V is the set of all nodes. The second design is to concatenate the graph representation in the three low-level MoE models. The second design performs better in our experiments, since the hidden representations are more expressive than the input features. Nonetheless, when we use a sparse MoE where only one or two experts are selected, the first design will be more memory efficient. It can determine the expert assignment before the computation of three expert models, thus reducing unnecessary computation. However, the best-performing method is to utilize all the experts, and in this case, the two designs are similar in efficiency. We utilize all the experts in our main experiments. We also apply the regularization term for the high-level gating network.

Two-Stage Training Strategy

Optimizing the hierarchical MoE is challenging. Different from previous MoE studies where expert networks have the same structures, our three low-level MoEs are very different and thus have different convergence speeds. The graph MoE model converges the fastest, since its MoE operates on the graph representation and has the least computation. Thus, the high-level gating network suffers more severely from expert polarization. It learns to assign nearly all the weights to the graph MoE model. If we simply increase the weight of the regularization term, the high-level gating network will learn to assign about $\frac{1}{3}$ weight to each expert, but the graph MoE model still converges the fastest. As a result, the graph MoE model will learn to output three times the prediction, while node and block MoE models will learn to output zero.

To address this unique challenge, we design a two-stage training strategy to encourage every expert model to perform well. In the first stage containing T epochs (warmup), we train the three expert models individually. In the second stage, we take turns training the whole model end-to-end and the three expert models individually. If we denote the label as Y, the prediction made by the i-th expert model as $\hat{Y_i}$, and the MSE loss function as $L(Y, \hat{Y_i})$, then we define the loss function at epoch t as:

$$L = \begin{cases} \frac{1}{3}[L(Y,\hat{Y}_1) + L(Y,\hat{Y}_2) + L(Y,\hat{Y}_3)] + \alpha L_R,\ if\ t < T\ or\ 2 \mid t \\ L(Y, \sum_{i=1}^{3} g_i \cdot \hat{Y}_i) + \alpha L_R + \beta L_{Rh},\ otherwise. \end{cases} \quad (6)$$

Here, $L_R = \frac{1}{3}(L_{R1} + L_{R2} + L_{R3})$, where L_{Ri} is the regularization term of the i-th low-level MoE. L_{Rh} is the regularization term of the high-level MoE. g_i is the weight assigned by the high-level gating network.

This two-stage training strategy does not increase the training time and is easy to implement. We only need to disable the high-level gating network and change the loss function in certain epochs. This strategy is only used during training. When we fine-tune a trained model on target kernels, we directly use the end-to-end joint training, since all the experts can already perform well and there is no risk of polarization. Besides, we initialize the high-level gating network to assign the same weights to the expert models, which can further prevent expert polarization. For low-level MoEs, we use the normal random initialization.

Experiments

Experiment Settings

Datasets. We use one of the most comprehensive benchmark datasets, HLSyn (Bai et al. 2023). It contains 42 kernels covering various categories: linear algebra of vectors and matrices, data mining, stencil operations, etc. We utilize the AMD/Xilinx HLS tool, Vitis 2021.1 (AMD/Xilinx 2020), to run HLS targeting the Xilinx Alveo U200 FPGA with a working frequency of 250MHz. We select 6 kernels as the target kernels that span representative categories including linear algebra, data mining, and stencil. The other kernels are source kernels. Table 2 shows the dataset statistics. We introduce their details in the appendix. The HLSyn dataset was generated by running the heuristics method, AutoDSE (Sohrabizadeh et al. 2021), for 24 hours. Many designs explored by AutoDSE are unavailable in HLS, so we collect this information to train a HARP classification model. Since its accuracy already exceeds 95%, there is no need to employ MoE on the classification model. We use the available designs in the dataset to train the regression model, and we employ MoE on the regression model.

Models. Based on our pre-explorations, we use 4 experts in the low-level MoEs, and we set the regularization terms' weights of both low-level and high-level MoEs to 5e-3. Our

		Offline evaluation	Online evaluation (FPGA speedup compared to AutoDSE)							
MoE category	Model	Total MSE	Fd	Gemv	Sy	Gemm	Ja	Tr	Average	Geo mean
No MoE	HARP	0.202±0.013	1.03	1.29	1	1	**1.08**	1.18	1.10	1.09
	HARP+MAML	0.732±0.167	1	1	1	1	1	1	1	1
	ProgSG	0.486±0.059	1	1	1	1	1	1.22	1.04	1.03
Single MoE	Node MoE	0.160±0.035	1.13	1.03	1.00	1	1	1.15	1.05	1.05
	Block MoE	0.171±0.019	1.43	1.11	1	1	1	1.07	1.10	1.09
	Graph MoE	0.216±0.046	1	1.28	**1.01**	1	**1.08**	**1.33**	1.12	1.11
Hierarchical MoE		**0.143±0.028**	**3.85**	**1.46**	1	**1.01**	1	1.23	**1.59**	**1.38**

Table 1: Domain generalization performance. "Geo mean" is the geometric mean speedup.

	Source	Fd	Gemv	Sy	Gemm	Ja	Tr
#Class	37697	418	428	526	1421	1837	1651
#Regre	9418	77	231	121	348	257	103

Table 2: Dataset statistics. "#Class" and "#Regre" denote the number of classification and regression data. The kernel acronyms represent "fdtd-2d-large", "gemver-medium", "syr2k", "gemm-p", "jacobi-2d", and "trmm-opt".

baselines include the SOTA GNN model, HARP (Sohrabizadeh et al. 2023); HARP+MAML (Bai et al. 2022), which applies meta-learning method MAML (Finn, Abbeel, and Levine 2017) on HARP inspired by (Bai et al. 2022) to learn a more generalizable initialization of parameters; and ProgSG (Qin et al. 2024), which combines HARP and language model. Although hierarchical MoE brings many more parameters, it needs about 32 GB of memory and can still fit into a single GPU. HARP and our model contain 359,370 and 1,329,403 parameters respectively.

Evaluation. We train the regression and classification models on the source kernels. We want to mimic the domain transfer situation of having scarce but representative labeled data on the target kernels, so we use 50 data points per kernel to fine-tune our regression model, and roughly the same ratio of data points, 265 samples per kernel, to fine-tune the classification model. To select representative data points, we use K-means based on the graph representation. We conduct both offline and online evaluations. In offline evaluation, we calculate the fine-tuned regression model's mean squared error (MSE) on the left-out data points in the target kernels for each regression objective and sum up the five objectives' MSE. In the online evaluation, we use the DFS search used in previous studies (Sohrabizadeh et al. 2022, 2023) to search for pragma designs. We limit our search range to 75,000 pragma designs, since it typically takes an hour. We use the fine-tuned classification model to predict the validity and the fine-tuned regression model to predict the latency and resource utilization. From designs that are predicted to be valid and satisfy resource constraints, we choose the top 10 designs with the lowest predicted latencies to run HLS. We report the best design from the selected top-10 and the training dataset of the target kernels during fine-tuning, since these are all labeled data points, and we calculate its speedup compared to the best design in the dataset. We run the offline evaluation five times and the online evaluation three times, then we report the mean results.

	HARP	HARP+MAML	Hierarchical MoE
MSE	0.426	3.227	**0.401**

Table 3: Offline evaluation results on more complex kernels.

Experiment Results

Table 1 shows the main results. In the main experiments, we use the second design of the high-level gating network as it performs better. MAML could not perform well in this setting. It might be because we have many more source kernels compared to the paper that proposes to use MAML on this task (Bai et al. 2022). Different kernels might result in different directions of the meta gradient, leading to unstable training. ProgSG which combines GNN and language models has a strong ability when the training data is sufficient, but it overfits in the data-scarce setting. During finetuning, the training loss is lower than 0.05, but the test loss is high. Comparatively, the hierarchical MoE is more generalizable. In the online evaluation, on most kernels, the best of three single MoE models outperforms HARP. However, different kernels favor different low-level MoEs. For example, the block is the best granularity for "Fd", while the graph is the best granularity for "Gemv", "Ja", and "Tr". By aggregating them together, hierarchical MoE performs the best or close to the best on almost every kernel.

Complex kernels. Kernels in the HLSyn dataset are generally small. We further test the models on 5 more complex kernels, including "3d-rendering" and "spam-filter" from the Rosetta benchmark (Zhou et al. 2018) and three self-constructed kernels, which are introduced in the appendix. They are closer to real-world applications. Following previous settings, we train the regression model on all HLSyn kernels and finetune it on 50 points per kernel on the new kernels. The results are shown in Table 3. It is challenging for all models to perform well, while the hierarchical MoE model performs the best. It remains a future work to generally enhance model capacity on complex kernels.

Ablation Study

To verify the effectiveness of our model design, we run extensive experiments on various model structures as abla-

Data split	N+B	N+G	B+G	N+B+G	Hierarchy
K-means	0.341	0.213	0.188	0.174	**0.143**
Random	0.893	0.562	3.126	0.984	**0.452**

Table 4: Using MoE on various granularities in a single model. "N", "B", and "G" represent node, block, and graph.

Metric	Node	Block	Graph	All
Offline (MSE)	0.164	0.177	0.279	**0.143**
Online (geo mean speedup)	1.34	1.13	1.35	**1.38**

Table 5: Using MoE on a single granularity in the hierarchical MoE. "Node/Block/Graph" means only using MoE on the node/block/graph granularity in the hierarchical MoE.

tion studies. First, instead of aggregating the three low-level MoEs, can we apply MoE on the three granularities together in a single model? We specify the detailed structure design in the appendix. Table 4 shows the results. When we use MoE on two or three granularities in a single model, the loss is usually higher than the lowest loss when only using MoE on one granularity. Apart from selecting 50 representative data points by K-means for fine-tuning, we also experiment with random data split. We use the same random split for all models to ensure fair comparison. Hierarchical MoE is the most stable model when it faces low-quality fine-tuning data in the random split. If we stack MoE on various granularities in the same model, the structure might be too complex and thus unstable to train on a small dataset.

Second, does the hierarchical MoE benefit from combining the three granularities? To answer it, we still use the hierarchical MoE structure, but we use the same low-level MoE model as the three experts of the high-level MoE. Table 5 shows the results. It verifies that aggregating the three granularities performs the best. Using only the node MoE or graph MoE also performs well in the online evaluation, and this might be due to the increased number of experts. Nonetheless, aggregating the three granularities could further improve the performance.

Third, we conduct an ablation study on the two-stage training and the high-level gating network's design, and the results are listed in Table 6. There are two components in the two-stage training: the warmup epochs, and training three experts jointly and separately in turn after the warmup. We either disable the alternative training or the warmup. The lowest MSE is achieved when we use both of them. Also, the high-level gating network based on hidden representations is better than that based on input features.

More ablation studies are in the appendix. They verify that the hierarchical MoE's performance is not due to increased parameter size or expert number, but the hierarchical structure and combination of three granularities.

	Two-stage training		W/o alternative train		W/o warmup	
Gating	Input	Hidden	Input	Hidden	Input	Hidden
MSE	0.149	**0.143**	0.193	0.159	0.152	0.184

Table 6: Ablation study of two-stage training and high-level gating network. "Input" is the first design of the high-level gating network, while "Hidden" is the second design.

Expert	Fd	Gemv	Sy	Gemm	Ja	Tr
Node MoE	37%	29%	36%	37%	**46%**	27%
Block MoE	**49%**	28%	26%	**40%**	32%	**37%**
Graph MoE	14%	**43%**	**38%**	23%	22%	36%

Table 7: Average assigned weights of the high-level MoE.

Expert	No pragma	Loop tiling	Pipeline	0<Parallel≤4	Parallel>4
1	**32%**	6%	6%	4%	2%
2	18%	8%	13%	41%	**81%**
3	22%	**64%**	**60%**	**43%**	8%
4	28%	22%	21%	12%	10%

Table 8: Average assigned weights of the block MoE. Each column shows the expert weights for pseudo nodes modified by a certain pragma type.

Analysis of Expert Assignment

We want to unveil the mystery of the gating networks. We show the average assigned weights by the high-level gating network in Table 7. According to Table 1, "Fd", "Gemv" and "Tr" have a strong preference for a certain granularity, while the other three kernels do not. Among them, the block MoE performs the best for "Fd", and it is also assigned the highest weight; graph MoE is the best expert for "Gemv" and "Tr", and it is also assigned the highest or nearly the highest weight. The weights are partially explainable.

We pick the best hierarchical MoE model based on the previous five repeated offline evaluation experiments to do a case study on the low-level MoE. Due to the limited space, here we only analyze the block MoE, and we analyze the node and graph MoEs in the appendix. The pragmas modify the pseudo nodes, so block MoE is a window for us to analyze the pragmas. We summarize the weights of each expert for each pragma type in Table 8. There are three types of pragmas: loop tiling, pipeline, and parallelization. The third expert is good at dealing with loop tiling, pipeline, and small parallel factors; the second expert is good at dealing with large parallel factors; the other two experts deal more with pseudo nodes that do not have pragmas. Different experts diversify their roles, which improves generalizability.

Conclusion

Domain generalization is a big challenge for HLS prediction models. Based on the unique challenges and opportunities, we propose the hierarchical MoE structure. In the low-level MoE, we apply MoE on one of the three natural granularities of the graph: node, basic block, or graph. In the high-level MoE, we aggregate the three low-level MoE models, so that different data points can flexibly decide which one to use. To address the severe expert polarization, we propose a two-stage training strategy. Extensive experiments have verified its effectiveness. Nonetheless, the generalizability of HLS prediction models still remains a big challenge, and future works can further improve the performance.

Acknowledgments

This work was partially supported by NSF grants 2211557, 1937599, 2119643, 2303037, and 2312501, SRC JUMP 2.0 PRISM Center, Amazon Research, Snapchat, and the CDSC industrial partners (https://cdsc.ucla.edu/partners/). The authors would also like to thank AMD/Xilinx for HACC equipment donation, and Marci Baun for editing the paper. J. Cong has a financial interest in AMD.

References

AMD/Xilinx. 2020. Vitis HLS. https://docs.xilinx.com/v/u/2020.2-English/ ug1416-vitis-documentation.

Bai, Y.; Sohrabizadeh, A.; Qin, Z.; Hu, Z.; Sun, Y.; and Cong, J. 2023. Towards a Comprehensive Benchmark for High-Level Synthesis Targeted to FPGAs. *Advances in Neural Information Processing Systems*, 36: 45288–45299.

Bai, Y.; Sohrabizadeh, A.; Sun, Y.; and Cong, J. 2022. Improving GNN-based accelerator design automation with meta learning. In *Proceedings of the 59th ACM/IEEE Design Automation Conference*, DAC '22, 1347–1350. Association for Computing Machinery. ISBN 9781450391429.

Cong, J.; Lau, J.; Liu, G.; Neuendorffer, S.; Pan, P.; Vissers, K.; and Zhang, Z. 2022. FPGA HLS today: successes, challenges, and opportunities. *ACM Transactions on Reconfigurable Technology and Systems (TRETS)*, 15(4): 1–42.

Cong, J.; Liu, B.; Neuendorffer, S.; Noguera, J.; Vissers, K.; and Zhang, Z. 2011. High-Level Synthesis for FPGAs: From Prototyping to Deployment. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 30(4): 473–491.

Cummins, C.; Fisches, Z. V.; Ben-Nun, T.; Hoefler, T.; O'Boyle, M. F.; and Leather, H. 2021. ProGraML: A graph-based program representation for data flow analysis and compiler optimizations. In *International Conference on Machine Learning*, 2244–2253. PMLR.

Dai, Q.; Shen, X.; Wu, X.; and Wang, D. 2019. Network Transfer Learning via Adversarial Domain Adaptation with Graph Convolution. *CoRR*, abs/1909.01541.

Ferretti, L.; Kwon, J.; Ansaloni, G.; Guglielmo, G. D.; Carloni, L. P.; and Pozzi, L. 2020. Leveraging Prior Knowledge for Effective Design-Space Exploration in High-Level Synthesis. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 39(11): 3736–3747.

Finn, C.; Abbeel, P.; and Levine, S. 2017. Model-Agnostic Meta-Learning for Fast Adaptation of Deep Networks. arXiv:1703.03400.

Han, H.; Li, J.; Huang, W.; Tang, X.; Lu, H.; Luo, C.; Liu, H.; and Tang, J. 2024. Node-wise Filtering in Graph Neural Networks: A Mixture of Experts Approach. *arXiv preprint arXiv:2406.03464*.

Hu, F.; Wang, L.; Wu, S.; Wang, L.; and Tan, T. 2021. Graph classification by mixture of diverse experts. *arXiv preprint arXiv:2103.15622*.

Jacobs, R.; Jordan, M.; Nowlan, S.; and Hinton, G. 1991. Adaptive Mixture of Local Expert. *Neural Computation*, 3: 78–88.

Kwon, J.; and Carloni, L. P. 2020. Transfer Learning for Design-Space Exploration with High-Level Synthesis. In *2020 ACM/IEEE 2nd Workshop on Machine Learning for CAD (MLCAD)*, 163–168.

Li, B.; Shen, Y.; Yang, J.; Wang, Y.; Ren, J.; Che, T.; Zhang, J.; and Liu, Z. 2023. Sparse Mixture-of-Experts are Domain Generalizable Learners. arXiv:2206.04046.

Liu, Y.; Ao, X.; Feng, F.; Ma, Y.; Li, K.; Chua, T.-S.; and He, Q. 2023a. FLOOD: A flexible invariant learning framework for out-of-distribution generalization on graphs. In *Proceedings of the 29th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, 1548–1558.

Liu, Z.; Zhang, C.; Tian, Y.; Zhang, E.; Huang, C.; Ye, Y.; and Zhang, C. 2023b. Fair graph representation learning via diverse mixture-of-experts. In *Proceedings of the ACM Web Conference 2023*, 28–38.

Pouget, S.; Pouchet, L.-N.; and Cong, J. 2024a. Automatic Hardware Pragma Insertion in High-Level Synthesis: A Non-Linear Programming Approach. arXiv:2405.12304.

Pouget, S.; Pouchet, L.-N.; and Cong, J. 2024b. Enhancing High-Level Synthesis with Automated Pragma Insertion and Code Transformation Framework. arXiv:2405.03058.

Qin, Z.; Bai, Y.; Sohrabizadeh, A.; Ding, Z.; Hu, Z.; Sun, Y.; and Cong, J. 2024. Cross-Modality Program Representation Learning for Electronic Design Automation with High-Level Synthesis. arXiv:2406.09606.

Schafer, B. C.; and Wang, Z. 2019. High-level synthesis design space exploration: Past, present, and future. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 39(10): 2628–2639.

Shazeer, N.; Mirhoseini, A.; Maziarz, K.; Davis, A.; Le, Q. V.; Hinton, G. E.; and Dean, J. 2017. Outrageously Large Neural Networks: The Sparsely-Gated Mixture-of-Experts Layer. *CoRR*, abs/1701.06538.

Shen, X.; Pan, S.; Choi, K.-S.; and Zhou, X. 2023. Domain-adaptive message passing graph neural network. *Neural Networks*, 164: 439–454.

Sohrabizadeh, A.; Bai, Y.; Sun, Y.; and Cong, J. 2022. Automated Accelerator Optimization Aided by Graph Neural Networks. In *2022 59th ACM/IEEE Design Automation Conference (DAC)*.

Sohrabizadeh, A.; Bai, Y.; Sun, Y.; and Cong, J. 2023. Robust GNN-based representation learning for HLS. In *2023 IEEE/ACM International Conference on Computer Aided Design (ICCAD)*, 1–9. IEEE.

Sohrabizadeh, A.; Yu, C. H.; Gao, M.; and Cong, J. 2021. AutoDSE: Enabling Software Programmers to Design Efficient FPGA Accelerators. arXiv:2009.14381.

Ustun, E.; Deng, C.; Pal, D.; Li, Z.; and Zhang, Z. 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.

Wang, H.; Jiang, Z.; You, Y.; Han, Y.; Liu, G.; Srinivasa, J.; Kompella, R. R.; and Wang, Z. 2023. Graph Mixture of Experts: Learning on Large-Scale Graphs with Explicit Diversity Modeling. arXiv:2304.02806.

Wu, M.; Pan, S.; Zhou, C.; Chang, X.; and Zhu, X. 2020. Unsupervised Domain Adaptive Graph Convolutional Networks.

Wu, N.; Xie, Y.; and Hao, C. 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. IEEE.

Wu, N.; Xie, Y.; and Hao, C. 2023. IronMan-Pro: Multi-objective 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): 900–913.

Wu, N.; Yang, H.; Xie, Y.; Li, P.; and Hao, C. 2022a. High-level synthesis performance prediction using GNNs: benchmarking, modeling, and advancing. In *Proceedings of the 59th ACM/IEEE Design Automation Conference*, DAC '22, 49–54. New York, NY, USA: Association for Computing Machinery. ISBN 9781450391429.

Wu, Q.; Zhang, H.; Yan, J.; and Wipf, D. 2022b. Handling Distribution Shifts on Graphs: An Invariance Perspective. In *International Conference on Learning Representations (ICLR)*.

Wu, S.; Cao, K.; Ribeiro, B.; Zou, J.; and Leskovec, J. 2024. GraphMETRO: Mitigating Complex Graph Distribution Shifts via Mixture of Aligned Experts. arXiv:2312.04693.

Zhang, Y.; Song, G.; Du, L.; Yang, S.; and Jin, Y. 2019. DANE: Domain Adaptive Network Embedding. In *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI-19*, 4362–4368. International Joint Conferences on Artificial Intelligence Organization.

Zhong, T.; Chi, Z.; Gu, L.; Wang, Y.; Yu, Y.; and Tang, J. 2023. Meta-DMoE: Adapting to Domain Shift by Meta-Distillation from Mixture-of-Experts. arXiv:2210.03885.

Zhou, Y.; Gupta, U.; Dai, S.; Zhao, R.; Srivastava, N.; Jin, H.; Featherston, J.; Lai, Y.-H.; Liu, G.; Velasquez, G. A.; Wang, W.; and Zhang, Z. 2018. Rosetta: A Realistic High-Level Synthesis Benchmark Suite for Software Programmable FPGAs. In *Proceedings of the 2018 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, FPGA '18, 269–278. New York, NY, USA: Association for Computing Machinery. ISBN 9781450356145.