Improving GNN-Based Accelerator Design Automation with Meta Learning

Yunsheng Bai, Atefeh Sohrabizadeh, Yizhou Sun, and Jason Cong
Computer Science Department, University of California - Los Angeles, USA
Los Angeles, CA, USA
{yba,atefehsz,yzsun,cong}@cs.ucla.edu

Abstract

Recently, there is a growing interest in developing learning-based models as a surrogate of the High-Level Synthesis (HLS) tools, where the key objective is rapid prediction of the quality of a candidate HLS design for automated design space exploration (DSE). Training is usually conducted on a given set of computation kernels (or kernels in short) needed for hardware acceleration. However, the model must also perform well on new kernels. The discrepancy between the training set and new kernels, called domain shift, frequently leads to model accuracy drop which in turn negatively impact the DSE performance. In this paper, we investigate the possibility of adapting an existing meta-learning approach, named MAML, to the task of design quality prediction. Experiments show the MAML-enhanced model outperforms a simple baseline based on fine tuning in terms of both offline evaluation on hold-out test sets and online evaluation for DSE speedup results[1].

ACM Reference Format:
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 (DAC) (DAC '22), July 10–14, 2022, San Francisco, CA, USA.* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3489517.3530409

1 Introduction

HLS aims to ease the design efforts for FPGA programming by raising the abstraction level. Nevertheless, a designer must explore more design candidates to achieve the optimal micro-architecture since the combination of different pragmas (compiler directives) create a larger solution space. A promising solution is to automate DSE using a deep learning based performance model that mimics the HLS tool [7, 14]. Such a model relies on training data, collected from a set of FPGA accelerator kernels, in the form of designs labeled with their performance metrics such as latency and resource usage. Once trained, models such as GNN-DSE [14] can be used to predict the quality of designs either from the same set of training kernels, or similar unseen kernels. A *domain shift* happens when the unseen kernel is very different from the kernels used for training, which may result in a significant accuracy drop.

In this paper, we investigate the possibility of adopting ideas from meta-learning (or learning-to-learn [16]) to address the domain shift issue. The objective of meta-learning is to obtain a model that can eventually generalize across many tasks with good data and computation efficiency [4]. For example, the K-shot image classification has been extensively studied recently [2], where the goal is to learn a classification model that can quickly adapt to a new class based on only K images from that class. In our case, we treat each kernel as a task. Thus, the model is trained on a set of kernels, and given a new kernel, we let the trained model adapt to only K designs of that kernel.

Following the K-shot framework, we formalize our problem as a K-shot regression problem. Given a training dataset $\mathcal{D}^{(\text{train})} = \{(X_i, Y_i)\}_{i=1}^{N}$, we wish to learn a model $Y = f_\theta(X)$ with a good initialization parameter θ that can quickly adapt to a new kernel, where K data points are provided, denoted as $\mathcal{D}^{(\text{new}),\text{K}} = \{(X_k^{(new)}, Y_k^{(new)})\}_{k=1}^{K}$. We further want to have a good prediction power on other test data points of the new kernel.

Among different meta-learning approaches, optimization-based methods such as MAML [3] and Reptile [9] are attractive to us as they are model-agnostic, i.e., they are **not** restricted to a certain type of learning model. Different from the traditional training scheme where the entire training dataset $\mathcal{D}^{(train)}$ is used to learn the model parameter θ, we mimic the setting where only K examples will be shown for a new kernel. In this paper, we show how MAML can be adapted to our task of a K-shot design quality prediction. In addition, at a higher level, we propose a new training-adaptation-evaluation workflow for learning-based accelerator design automation. It has the advantage of quickly adapting a learning model to any new kernel that is unseen during training. We experimentally evaluate the efficacy of the proposed approach using a recent model, GNN-DSE, as an example, and name our MAML-enhanced version as GNN-DSE-MAML.

In summary, our contributions can be summarized as follows:

- We propose a new workflow to train and adapt a learning-based DSE model for automated accelerator optimization based on meta-learning to address the domain shift problem.
- We adapt the MAML algorithm to GNN-DSE, specifically, in the training and adaptation stages, to obtain a trained model, named GNN-DSE-MAML, that can be quickly adapted to different kernels.
- The experimental results demonstrate adaptation is necessary for a trained GNN-DSE model to perform well on unseen kernels with a domain shift, and that MAML leads to higher accuracy and speedup results on such kernels under 3 out of 5 cases.

2 Background and Related Work

2.1 Learning-based DSE

Despite the recent success of domain-specific accelerators over general-purpose CPUs (e.g., [5]), quickly obtaining the optimal design for a particular kernel remains a challenge. This is due to not only the time-consuming evaluation of the designs using the commercial tools, but also the large combinatorial search space of a given kernel consisting of the different design candidates. The former challenge has recently been tackled with modern machine learning and deep learning techniques [7, 14] to assess a design's

[1]This work is supported by the CAPA award jointly funded by NSF (CCF-1723773) and Intel (36888881), and the RTML award funded by NSF (CCF-1937599).

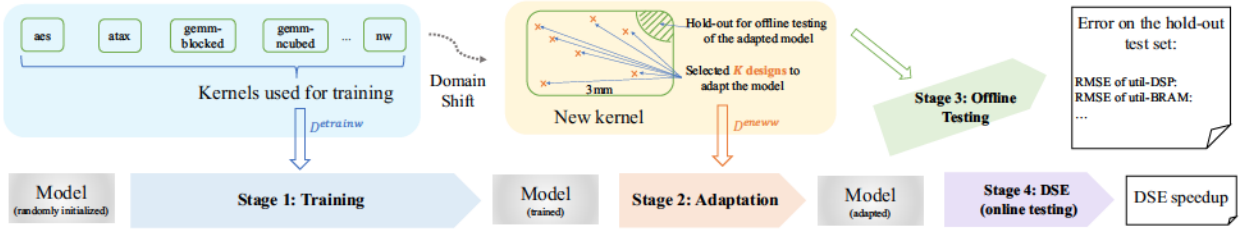

Figure 1: The workflow of GNN-DSE-MAML.

Algorithm 1 Training procedure of GNN-DSE-MAML

Require: $p(\mathcal{P}^{(train)})$: distribution over kernels (programs) for training
Require: α, β: step size hyperparameters
1: randomly initialize θ
2: **while** not done **do**
3: Sample batch of kernels $\mathcal{P}_i \sim p(\mathcal{P}^{(train)})$
4: **for all** $\mathcal{P}_i$ **do**
5: Sample K datapoints $\mathcal{D} = \{X_j, Y_j\}$ from $\mathcal{P}_i$
6: Evaluate $\nabla_\theta \mathcal{L}_{\mathcal{P}_i}(f_\theta)$ using $\mathcal{D}$ and $\mathcal{L}_{\mathcal{P}_i}$ in Equation 1
7: Compute adapted parameters with gradient descent: $\theta_i' = \theta - \alpha \nabla_\theta \mathcal{L}_{\mathcal{P}_i}(f_\theta)$
8: Sample datapoints $\mathcal{D}_i' = \{\mathbf{x}^{(j)}, \mathbf{y}^{(j)}\}$ from $\mathcal{P}_i$ for the meta-update
9: **end for**
10: Update $\theta \leftarrow \theta - \beta \nabla_\theta \sum_{\mathcal{P}_i \sim p(\mathcal{P})} \mathcal{L}_{\mathcal{P}_i}(f_{\theta_i'})$ using each $\mathcal{D}_i'$ and $\mathcal{L}_{\mathcal{P}_i}$ in Equation 1
11: **end while**

quality in milliseconds. We use our prior work, GNN-DSE [14], to demonstrate the proposed techniques for addressing the domain shift in this paper. The core component of GNN-DSE is a Graph Neural Network (GNN)-based model for predicting the quality of a design. The goal of the model, $Y = f_\theta(X)$, is to make predictions as accurate as possible so that the DSE process can explore more design points by replacing the time-consuming HLS tool. However, quickly adapting a trained model to new kernels has yet to be researched.

2.2 Meta-Learning for domain shift

The problem of adapting machine learning models across tasks and domains has long been studied in computer vision [12], natural language processing [8], graph-related tasks [18], etc. Here, we focus on our specific task of the K-shot design quality regression, and discuss exsiting approaches to deal with the distributional shift across kernels.

One simple approach is to apply a relatively simple technique known as fine-tuning. As its name suggests, one can further tune $f_\theta(X)$ by performing several steps of the gradient update, e.g., back-propagation for neural network models such as GNN-DSE, on several labeled design points of that new kernel. Since the input kernels are represented the same way across different kernels, and the output regression targets bear the same meanings, no new learning component is needed for our task.

Despite the simplicity of such a fine-tuning method, more advanced methods are shown to be more effective to address the domain shift problems [19, 21]. For that purpose, we adopt the Model-Agnostic Meta-Learning (MAML) [3] (presented in Algorithm 1) approach proposed in the meta learning community. As mentioned previously, meta learning deals with the higher level problem of learning to learn, where we utilize one of its most popular tasks, K-shot prediction problem, to formulate our problem. Numerous approaches have been proposed through the years [4]. One category, for example, is based on metric-learning, where the model tries to match the new K data points with some existing data points in the training set [17]. In this work, however, we turn to the optimization approach such as MAML, which can be applied to any learning-based $f_\theta(X)$, including GNN-DSE.

3 Our Proposed Methodology

Fig. 1 depicts a high-level diagram of GNN-DSE-MAML which operates in four stages: training, adaptation, offline testing, and DSE. We begin with a collection of kernels forming the training dataset, $\mathcal{D}^{(\text{train})}$. The randomly initialized GNN-DSE model is trained on $\mathcal{D}^{(\text{train})}$ for many epochs using the MAML algorithm. Then, the trained model is adapted to K designs that we sample from the new kernel. Finally, the adapted model is used as a surrogate to the HLS tool to run the offline testing and DSE stages on the new kernel, measuring the accuracy of the updated model on the hold-out test set and searching for the Pareto-optimal design points, respectively.

3.1 Training via Meta-Learning

The purpose of training GNN-DSE-MAML is to learn a general model parameter θ that can quickly adapt to new kernels. We adapt the original MAML algorithm for our task. To get started, we assume a learning-based model, f_θ, which takes a kernel X as input, and outputs a set of design quality metrics, such as latency, resource utilization, etc., which we denote as Y. We use GNN-DSE as an example to illustrate the training technique, but in practice, any model following the same setup can be used. The readers are referred to the original paper of GNN-DSE [14] for a more comprehensive description of the input, output formats, and the architecture. GNN-DSE performs the normal supervised training whose loss function takes the following form:

$$\mathcal{L}_{\mathcal{P}_i}(f_\phi) = \sum_{X_j, Y_j \sim \mathcal{P}^{(train)}} \|f_\phi(X_j) - Y_j\|_2^2, \quad (1)$$

However, GNN-DSE-MAML performs the following training within each epoch: First, a batch of N kernels are sampled (line 3); then MAML loops through each kernel, samples K labeled designs from the kernel (line 5), updates the model to obtain a temporary model with parameter θ_i' (line 7), and further samples another K designs from the kernel forming a validation set $\mathcal{D}_i'$ (line 8). At the end of the loop, MAM obtains N copies of the model together with N validation sets, each corresponding to the updated version of the model using one kernel.

The key step is to aggregate these N models together to produce the final updated model for this epoch (line 10). Specifically, the aggregation step (line 10) evaluates each updated model $f_{\theta_i'}$ on its earlier sampled validation set $\mathcal{D}_i'$ by computing the gradient $\nabla_\theta \sum_{\mathcal{P}_i \sim p(\mathcal{P})} \mathcal{L}_{\mathcal{P}_i}(f_{\theta_i'})$. The gradients from N models are accumulated to perform a final update of the original model with step size β. Intuitively, this ensures each updated model $f_{\theta_i'}$ has a chance to update the original model f_θ by being evaluated on another validation set $\mathcal{D}_i'$ that is different from the set used for training $\mathcal{D}$. This helps to reduce the chance of over-fitting $f_{\theta_i'}$ to $\mathcal{D}$. The readers are referred to the original paper of MAML[3] and various follow-up papers (.g., [1, 6, 11]) for a more detailed analysis and understanding of MAML.

3.2 Adapting the Trained Model for a New Kernel

The next stage of Gnn-Dse-Maml is to adapt the trained model for a new kernel, which is likely to be very different from the kernels used for training, causing the domain shift issue. The high-level idea here is to select as few design points as possible from the new kernel, since each design point requires running the time-consuming HLS tool to obtain its label for adapting the model.

There are two challenges in the adaptation stage. The first one is how to smartly select only K designs, where K is typically 10 or 20, out of the very large design space with many candidate designs as the labeled designs, $\mathcal{D}^{(\text{new}),\text{K}}$, for adaptation. We leave the topic as future direction and use random sampling in this work. The second challenge is how to adapt the model to these K new designs of the new kernel. Thankfully, MAML, by its design, allows for the easy adaptation of a trained model to K data points sampled from another kernel via lines 5-7 in Algorithm 1.

3.3 Testing the Adapted Model via Offline and Online Stages

The last two stages of Gnn-Dse-Maml are to test the efficacy of the adapted model on the new kernel. For the offline testing stage, we hold out a certain percentage of labeled designs from the new kernel, evaluate the adapted model on these designs in the same way as the original Gnn-Dse, and report the root mean-squared error (RMSE) on the design's objectives However, the ultimate goal is to find the Pareto-optimal design points. Therefore, we feed the adapted model to the DSE process (as described in the original Gnn-Dse paper) which essentially ranks the design points based on their quality predicted by the adapted model. It then evaluates the top-ranked design points using the HLS tool to determine their true objectives and whether they form the Pareto-optimal design points.

4 Evaluation

4.1 Experimental Setup

Gnn-Dse-Maml follows the same model architecture as Gnn-Dse, and to ensure fair comparison, we also use the same 9 training kernels as the original paper of Gnn-Dse, as detailed in Table 1, which includes kernels from the MachSuite [13] and the Polyhedral (Polybench) [20] benchmark suites. The database is generated employing both AutoDSE [15] and Gnn-Dse with the Xilinx Virtex Ultrascale+ VCU1525 as the target FPGA. Our model predicts the `latency` in the form of *cycle counts*, and the resource utilization in the forms of `DSP`, `BRAM`, `LUT`, and `FF`. All the models are trained using the PyTorch [10] library. All models are trained for 5000 iterations, and the best model with the lowest validation error is selected for the adaptation stage.

To demonstrate the ability of Gnn-Dse-Maml to adapt to different kernels, we choose the following 5 kernels from Polybench [20], which are new to Gnn-Dse and it does not perform well on them, suggesting a domain shift compared to the training set.

- jacobi-1d: Jacobi-style stencil computation over 1D data with 3-point stencil pattern.
- fdtd-2d: Simplified Finite-Difference Time-Domain method for 2D data consisting of three stencil operations.
- gemm: A scalar matrix multiplication followed by a matrix multiplication calculating $\beta C + \alpha A \cdot B$.
- 3mm: Three matrix multiplications calculating $(A \cdot B) \cdot (C \cdot D)$, which creates a solution space with more than 17 trillion design points.
- gemver: Multiple loop sections with vector-vector and matrix-vector multiplications.

Although we have one stencil operation in the training set, the stencil window size and the number of stencil operations are different compared to the ones in `jacobi-1d` and `fdtd-2d`. The last three kernels cause a domain shift due to the increase in the number of different operations compared to other matrix-vector operations of our training set. In other words, the Pareto-optimal design point of each operation separately may not create an overall Pareto-optimal design point. This is because not only increasing the number of operations restricts the on-chip resources for the others, these operations also share matrices/vectors which need consistent array partitioning.

A key difference between the set up of Gnn-Dse-Maml and Gnn-Dse is the split of the dataset into training and testing kernels. In Gnn-Dse, 80% of the labeled designs within each training kernel is used for training, while in Gnn-Dse-Maml, we test the ability of the model to adapt to different testing kernels. In order to evaluate the trained model on the 5 testing kernels, we have the following three settings for the offline and online evaluation stages:

- Gnn-Dse-Unadapted: The trained Gnn-Dse model.
- Gnn-Dse-FineTune: The trained Gnn-Dse model adapted to K sampled designs.
- Gnn-Dse-Maml: The model is trained using the MAML algorithm, and the trained model is adapted to K sampled designs.

To ensure a consistent comparison for both, Gnn-Dse-FineTune and Gnn-Dse-Maml, we run 5 gradient steps on the K designs.

4.2 Model Evaluation

We pre-process the dataset the same way as Gnn-Dse. For each one of the 5 new kernels, we take the trained model and perform adaptation by sampling $K = 20$ design points of that kernel and getting their labels by running the HLS tool. We argue that the choice of K should be small enough, since otherwise, it would take too much time to obtain the labels. For the offline evaluation stage though, we run AutoDSE [15] up to 20 hours to report the total root mean-squared error (RMSE) on a larger set of data (referred to as the hold-out set). The best design found by AutoDSE is also used as a baseline to measure the speedup of each of the models.

Table 1: Design space and the database of the 9 kernels used for training and the 5 kernels for testing. For each testing kernel, we restrict ourselves to using only 20 labeled designs to adapt the trained models.

Kernel	Training									Testing				
	aes	atax	gemm-blocked	gemm-ncubed	mvt	spmv-crs	spmv-ellpack	stencil	nw	jacobi-1d	fdtd-2d	gemm	3mm	gemver
# pragmas	3	5	9	7	8	3	3	7	6	5	16	8	21	13
Database Size (# Valid)	44	140	189	199	255	52	60	295	103	20	20	20	20	20

4.3 Results of Design Space Exploration

For testing our approach, we collected 5 new kernels that the unadapted model of GNN-DSE [14] was not successful in finding their Pareto-optimal design points. GNN-DSE has shown that it can perform well on new kernels with a similar domain to its training set. Therefore, when it fails to produce comparable results as those by AutoDSE, it is likely there is a domain shift that needs adaptation. Table 2 summarizes the performance of the unadapted model of GNN-DSE and the adapted models as explained in Section 4.1 when they have used in the DSE process for up to 1 hour. The results show that the MAML-based adaptation can achieve great performance for 3 of the new kernels. In fact, it can get a significant speedup for `3mm` compared to AutoDSE. The solution space of this kernel consists of more than 17 trillion design candidates that AutoDSE got to explore only 149 of them after 20 hours since it relies on the HLS tool for evaluating each candidate. However, the model-based DSE can explore about 80K candidates after a 1 hour search. This helps it to find a better design point given an accurate-enough model.

For two of the kernels, `ftdt-2d` and `gemver`, the MAML results lead to *Timed Out*, as the HLS tool cannot finish the synthesis after 3 hours. Manual inspection shows that both examples have multiple sections of nested loops, but with shared array variables. The MAML-based model uses high degree of parallelization for each section of the loop nests, requiring complex and extensive array partitioning of the shared array variables that overwhelm the HLS tool. Unfortunately, such cases were not covered in the K samples, highlighting the importance of sample selection, which will be further investigated in the future.

Table 2: The DSE speedup with respect to AutoDSE [15] after 20 hours.

Method	jacobi-1d	fdtd-2d	gemm	3mm	gemver
Gnn-Dse-Unadapted	0.44×	0.06×	0.87×	0.30×	0.20×
Gnn-Dse-FineTune	0.54×	0.04×	0.18×	1.00×	0.22×
Gnn-Dse-Maml	1.00×	TO	1.21×	64.52×	TO

TO: Timed Out

4.4 Results of Offline Testing

Table 3: Offline evaluation results. RMSE (the lower, the better) is used to evaluate each model's error on the hold-out test set of each new kernel.

Method	jacobi-1d	fdtd-2d	gemm	3mm	gemver
Gnn-Dse-Unadapted	4.2496	6.7047	7.5337	9.1584	4.4717
Gnn-Dse-FineTune	3.2611	4.0831	**1.7342**	6.2930	3.1600
Gnn-Dse-Maml	**2.3898**	**2.4912**	2.1116	**5.9670**	**3.0303**

We report the offline evaluation results in Table 3. The results show that adaptation is necessary for the unadapted model to obtain lower error on the hold-out sets, and comparing between fine-tuning and MAML, under 4 out of 5 kernels, MAML leads to a more accurate adapted model. This suggests the importance of training the model under the meta-learning framework, and the better accuracy of MAML can be attributed to its ability to provide a better model parameter θ.

5 Conclusion and Future Work

We investigate the use of meta-learning, in particular, the MAML algorithm, to enhance the training of deep learning based prediction models, such as Gnn-Dse, for HLS performance prediction and automated DSE. Our MAML based model Gnn-Dse-Maml shows advantages over both Gnn-Dse and its adaptation based on fine tuning when there is domain shift. Although Gnn-Dse-Maml produces promising results, it also reveals further research opportunities, especially on design sample selection of new kernels for MAML-based adaptation, and the possibility of further online adaptation during DSE.

References

[1] S. Arnold et al. 2021. When maml can adapt fast and how to assist when it cannot. In *International Conference on Artificial Intelligence and Statistics*. PMLR, 244–252.

[2] G. S. Dhillon et al. 2020. A baseline for few-shot image classification. *ICLR* (2020).

[3] C. Finn et al. 2017. Model-agnostic meta-learning for fast adaptation of deep networks. In *ICML*. PMLR, 1126–1135.

[4] T. Hospedales et al. 2020. Meta-learning in neural networks: A survey. *arXiv preprint arXiv:2004.05439* (2020).

[5] N. P. Jouppi et al. 2017. In-datacenter performance analysis of a tensor processing unit. In *Proceedings of the 44th annual international symposium on computer architecture*. 1–12.

[6] C. H. Kao et al. 2022. MAML is a Noisy Contrastive Learner in Classification. In *ICLR*.

[7] J. Kwon et al. 2020. Transfer Learning for Design-Space Exploration with High-Level Synthesis. In *2020 ACM/IEEE MLCAD*.

[8] E. Lekhtman et al. 2021. DILBERT: Customized Pre-Training for Domain Adaptation with Category Shift, with an Application to Aspect Extraction. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*. 219–230.

[9] A. Nichol et al. 2018. On first-order meta-learning algorithms. *arXiv preprint arXiv:1803.02999* (2018).

[10] A. Paszke et al. 2019. PyTorch: An Imperative Style, High-Performance Deep Learning Library. In *Advances in Neural Information Processing Systems 32*.

[11] A. Raghu et al. 2020. Rapid learning or feature reuse? towards understanding the effectiveness of maml. *ICLR* (2020).

[12] P. Z. Ramirez et al. 2019. Learning across tasks and domains. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 8110–8119.

[13] B. Reagen et al. 2014. Machsuite: Benchmarks for accelerator design and customized architectures. In *IISWC*.

[14] A. Sohrabizadeh et al. 2022. Automated Accelerator Optimization Aided by Graph Neural Networks. *DAC* (2022).

[15] A. Sohrabizadeh et al. 2020. AutoDSE: Enabling Software Programmers Design Efficient FPGA Accelerators. *arXiv preprint arXiv:2009.14381* (2020).

[16] S. Thrun et al. 1998. Learning to learn: Introduction and overview. In *Learning to learn*. Springer, 3–17.

[17] O. Vinyals et al. 2016. Matching networks for one shot learning. *NeurIPS* 29 (2016).

[18] G. Yehudai et al. 2021. From local structures to size generalization in graph neural networks. In *International Conference on Machine Learning*. PMLR, 11975–11986.

[19] M. Yi et al. 2021. Improved OOD Generalization via Adversarial Training and Pretraing. In *ICML*. PMLR, 11987–11997.

[20] T. Yuki et al. PolyBench/C. ([n. d.]). https://web.cse.ohio-state.edu/~pouchet.2/software/polybench/

[21] M. Zhang et al. 2021. Adaptive risk minimization: Learning to adapt to domain shift. *NeurIPS* 34 (2021).