2023 Design, Automation & Test in Europe Conference (DATE 2023)

AIRCHITECT: Automating Hardware Architecture and Mapping Optimization

Ananda Samajdar
Georgia Tech / IBM Research
anandsamajdar@gatech.edu

Jan Moritz Joseph
RWTH Aachen
Aachen, Germany

Tushar Krishna
Georgia Tech
Atlanta, GA, USA

***Abstract*—Design space exploration and optimization is an essential but iterative step in custom accelerator design involving costly search based method to extract maximum performance and energy efficiency. State-of-the-art methods employ data centric approaches to reduce the cost of each iteration but still rely on search algorithms to obtain the optima. This work proposes a learned, constant time optimizer that uses a custom recommendation network called AIRCHITECT, which is capable of learning the architecture design and mapping space with a 94.3% test accuracy, and predicting optimal configurations, which achieve on an average (GeoMean) 99.9% of the best possible performance on a test dataset with 10^5 GEMM (GEneral Matrix-matrix Multiplication) workloads.**

***Index Terms*—machine learning, design space exploration, architecture design, mapping optimization**

I. Introduction

The performance and energy efficiency gains of using custom accelerators come from the tight coupling of workload requirements and hardware design. Therefore, design space exploration (DSE) and optimization play an essential role in architecture development [12]. Moreover, given the interdependence of hardware performance with workload requirements, design refreshes are frequent and necessary with evolving workloads.

Challenge. Design optimization is a data-driven process and is conducted typically using search. Fig. 1 shows a schematic example of design optimization. In *Step* ①, for a given set of target workloads and design constraints, a subset of design points is identified and evaluated using a cost function (typically a simulator). In *Step* ②, the optimal configuration is determined among these points using some search algorithm. These steps are repeated for each unique set of workloads and design constraints, making the process expensive.

State-of-the-art. Existing techniques that use previously generated optimization data to reduce this cost can be broadly categorized into two sets. The first set of approaches estimate the performance or power cost of different microarchitecture choices using regression-based methods such as linear or spline functions [1], [4], SVM-based regressors [11], or neural-network-based regression models [5], [9]. The current state-of-the-art is AutoTVM [2] which uses a learned model to predict the cost of different execution configurations to speed up the search for optimal device placement. The second set of approaches speed up the search process by directed sampling using machine learning-based techniques like a genetic algorithm (GA) or reinforcement learning (RL) [7], [8], [15].

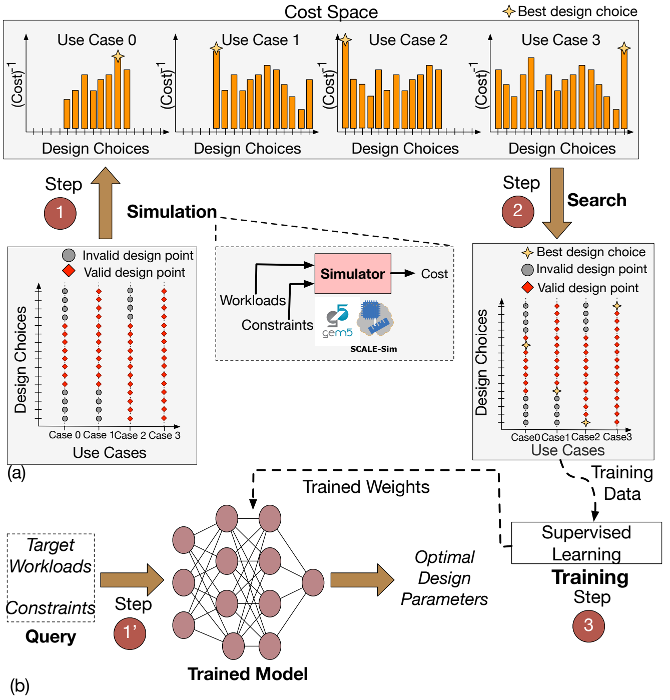

Fig. 1. (a) Conventional design space exploration and optimization pass, where iterative simulations (step 1) and search (step 2) are needed to find optimal design and mapping for each use case. (b) Proposed approach of using a learnt model for constant time optimization, which requires constant time query to the learnt model (step 1'), trained on previously generated simulation data (step 3)

Proposed method and contributions. *In this work, we propose a constant time optimization method*, using a trained recommendation model to predict the optimal hardware architecture and mapping at constant time. This approach eliminates the need for both simulations and searches. As depicted in Fig. 1(b), only one inference (*Step* ①’) is needed to get optimal parameters. We use the previously generated optimization data to train this model offline (*Step* ③) and learn the relationship between design requirements and optimal architecture and mapping parameters. As a proof of concept, we examine the design space and effectively apply our approach to three hardware architecture and mapping case studies in Sec. II. Thereby our contributions are:

– *We present systematic design-aware analysis (Sec. III) on*

This work is supported by NSF Award 1909900

978-3-9819263-7-8/DATE23/© 2023 EDAA

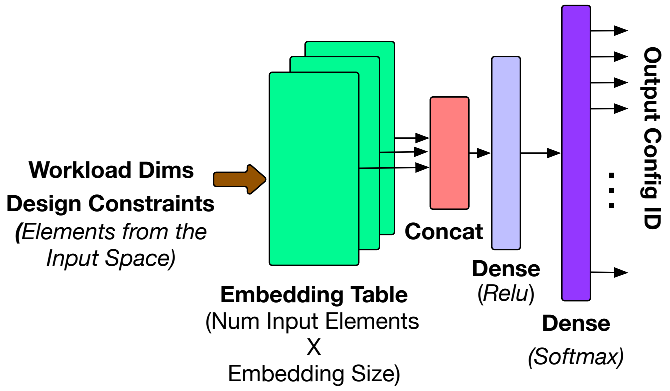

Fig. 2. General structure of the proposed class of recommendation neural networks (AIRCHITECT)

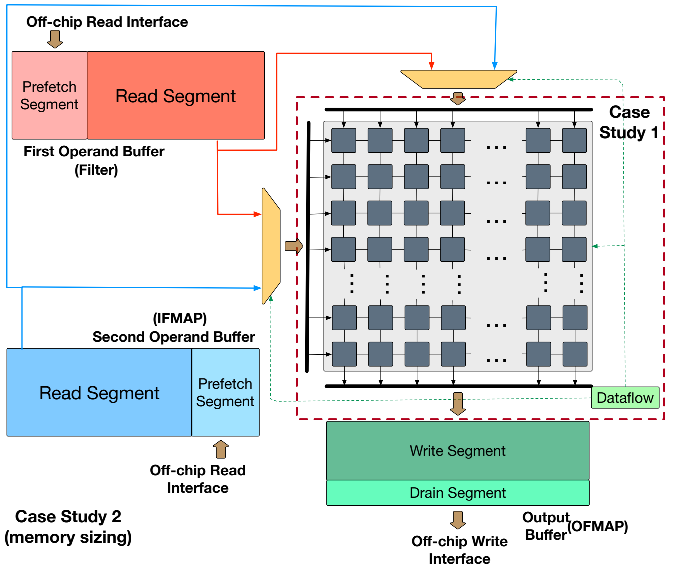

Fig. 3. Monolithic systolic array template with configurable array dimensions, choice of dataflow, and configurable buffer sizes

the design and mapping space of custom architectures via three case studies, demonstrating the feasibility of learning these spaces.

– We formulate traditional DSE tasks as ML classification/recommendation tasks by "quantizing" the optimization space and structuring the input and the output spaces (Sec. IV). This enables the conversion of optimization tasks to learning problems, solvable using existing ML classification models.

– We propose and evaluate AIRCHITECT (Fig. 2), a custom-designed neural recommendation network capable of learning the design space to a high degree of prediction accuracy (Sec. V). The configurations predicted by AIRCHITECT achieve 99.99% of the best achievable performance on a test data of 10^5 workloads in the case studies.

II. Case Studies

Background: Systolic Array. Systolic arrays are efficient for dense matrix multiplication and naturally have been adapted in many commercial and academic proposals for DNN acceleration. Fig. 3 depicts the schematic template of our design task.

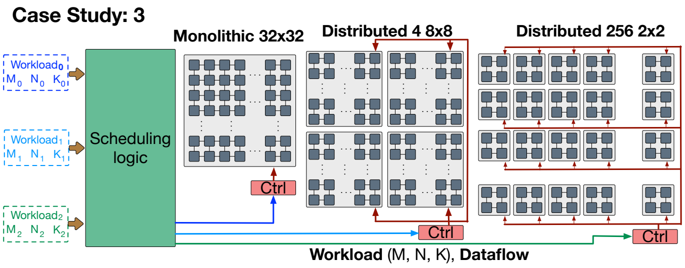

Fig. 4. Schematic view of the multi-array scheduling case study

Dataflow. There are three distinct strategies, called dataflows [3] for mapping compute onto the systolic array, namely Output Stationary (OS), Weight Stationary (WS), and Input Stationary (IS). Although many different dataflows exist for spatial arrays, we consider only true systolic dataflows that solely use neighbor-to-neighbor communication [14].

In this paper, we target three design-spaces / case studies:

– Case Study 1: Array and Dataflow Prediction. This case study focuses on the compute array in Fig. 3. The array has two sets of design parameters we wish to learn and recommend for a given workload (i.e., DNN layer): (i) array shape (choosing rows and columns), and (ii) dataflow. These design parameters, with the dimension of the GEMM workload determine how much of the array conducts useful work (utilization). A mismatch of workload dimension, array dimensions, and mapping scheme (dataflow) leads to a loss in performance and efficiency.

– Case Study 2: Buffer Size Prediction. Given a array of a fixed size and shape, this case study focuses on sizing its SRAM buffers. As shown in Fig. 3, the array has three buffers. Two of them prefetch and store the input operands (Input Feature map (IFMAP) and Filter in CNN terminology [3]),while the third buffer temporarily stores the generated output elements (Output feature map (OFMAP)) or partial sums. Proper sizing of the buffers leads to improved spatio-temporal reuse of operands, improving performance and lowering energy consumption for data movement.

– Case Study 3: Multi-array Scheduling. This case study schedules multiple independent layers on given heterogeneous arrays (each with different size and memory). Fig. 4 schematically shows heterogeneous arrays with one monolithic 32×32 array and two distributed configurations, 4 8×8 and 256 2×2 arrays. It is capable to serving three workloads simultaneously. The optimizer matches the three workloads to the arrays and finds the optimal dataflow such that execution time and energy consumption is minimized.

III. Design-aware Analysis

A. Array shape and dataflow

We examine the space of optimal array dimensions (row/-column) obtained in Fig. 5(a-c) when the design is constraint to 2^9 (i.e., the maximum number of MAC units). This chart is generated by searching the optimum for 10^4 GEMM workloads, by the sampling the operand dimensions (M, N, and K for multiplying matrices $M \times K$ and $K \times N$) from the distribution depicted in Fig. 7(a). In Fig. 5(a, b, c) we show the optimal array dimensions for workloads that yield the best runtime when

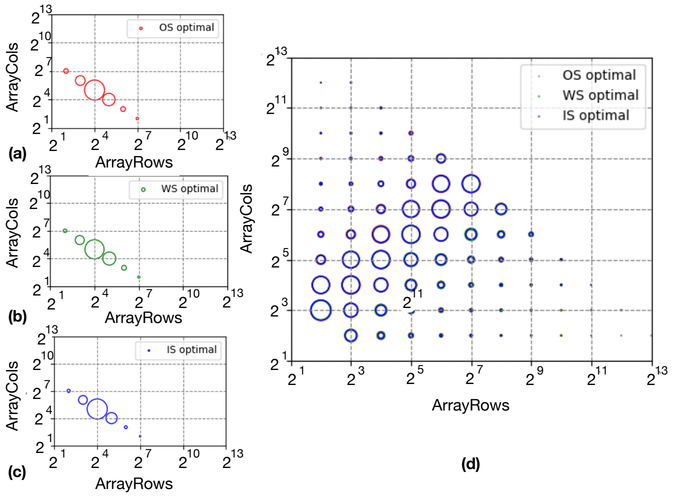

Fig. 5. Relative frequency of optimal array dimensions for GEMM workloads using (a) OS (b) WS, and (c) IS dataflow for 2^9 MAC unit limit. (d) The pattern of the optimal aspect ratio and the optimal dataflow for GEMM workloads on systolic arrays with 2^5 to 2^{15} MACs

using OS, WS, and IS dataflow. The radius of the marker at each array dimension (row/col) captures the relative frequency at which the said dimension is optimal.

In Fig. 5(d) we capture the relative frequency of optimal array dimensions for different values of maximum MAC units from 2^5 to 2^{15}. The following observations can be made from this figure. *First*, we see a clear pattern that the most frequent array configurations are near to a square shape or have twice as many columns as rows, unlike the conventional approach of choosing square configurations [6]. *Second*, all of the array configurations are optimal for at least one workload. Thus, no dimension can be summarily rejected from the search space to find the global optimum. *Third*, there is no apparent pattern for determining the optimal dataflow given the information about the optimal array configurations.

To understand the variation of the optimal predicted dataflow, we study the design space for the shape of the operand matrices in Fig. 6(a-c). In Fig. 6(a) the x-axis captures the ratio of height and width of the first operand (M:K).The y-axis captures all the possible dimensions of the arrays using MAC units from 2^5 to 2^{15} in powers of 2. Each data point represents the optimal array configuration and dataflow for a GEMM workload, where colors represent the dataflow. Fig. 6(b, c) depict similar distributions w.r.t. the shape of the second operand matrix ($K \times N$) and the output matrix ($M \times N$). One can observe from Fig. 6(a) that distinguishing between OS and WS is possible just from the shape of the first operand matrix. However, determining either over IS dataflow is difficult as there exists a significant overlap. Similarly, Fig. 6(c) depicts that the dimensions of the output operand matrix can help to choose between WS and IS, while Fig. 6(b) shows the same for IS over OS depending on the shape of the second operand.

B. SRAM Buffer Sizing

For design aware analysis for buffer size prediction, we consider the optimal buffer dimensions in a systolic array, obtained by searching for 10^4 distinct GEMM workloads running on randomly chosen array dimensions between 2^4 to 2^{18}, interface bandwidth between 1 to 100, and dataflow among OS, WS, and IS. For the sake of simplicity, we limit the search space to buffer sizes from 100KB to 900KB, changed in increments of 100KB. The objective of the search algorithm is to find the sizes of the three buffers simultaneously such that the overall stall cycles are minimized, and in case of multiple configurations with the same cost, the one with cumulatively minimum size.

Fig. 6(d-f) depict the correlation of the optimal buffer sizes with the input parameters, interface bandwidth, size of workload matrices, and dataflow. Fig. 6(d) shows for IS dataflow, small buffer sizes of IFMAP operand is optimal as this operand is most optimally reused in this scheme. Similarly, for WS dataflow, the filter operand favours small buffer dimensions (Fig. 6(e)). For generated buffer size however the optimal buffer sizing is somewhat counter-intuitive (Fig. 6(f)), as smaller output buffers are chosen when the size of the output operand increase. From our design standpoint, allocating larger buffer for input operands is beneficial for energy efficiency and performance. Larger output matrix sizes correlate with larger input matrices, which need larger input buffers; This leads to smaller output buffer sizes. For smaller matrices, a larger share is available for the output buffer.

C. Multi-array Scheduling

Fig. 6(g) depicts the space of optimal schedules when three arrays are used to run three distinct workloads concurrently as depicted in Fig. 4. The chart plots the optimal mappings in the space of the computation size of each workload. For ease of visualization, we plot three mappings chosen randomly out of the 162 possible. Similar to the previous case studies, clear cluster formations can be used to predict the optimal mapping from the GEMM workload parameters.

IV. Learning Architecture and Mapping Space

A. Design Optimization as Learning Problem

Empirical inspection of hyper-planes obtained as depicted in our previous analysis indicate that a classifier based method should work to predict optimal design and mapping parameters. A separate model needs to be constructed for each output parameters, which is trained such that it predicts the optimal value when queried with workload information and design constraints. However, independent classifiers for each feature tend to miss the interdependence of architecture and mapping parameters. We found recommendation models, which are capable of predicting multiple such parameters simultaneously are better suited in this case. In this setting, multiple parameters are clubbed into bins, and the trained model is expected to recommend the bin corresponding to the optimal parameters.

B. Dataset Generation

Case Study 1: Array and Dataflow prediction. For this case study, the *input space* comprises of four integers, three for the operand dimensions (M, N, and K), and one integer capturing the MAC unit budget. The dimension bounds are determined from layers of popular conv-nets as shown in Fig. 7(a). The MAC

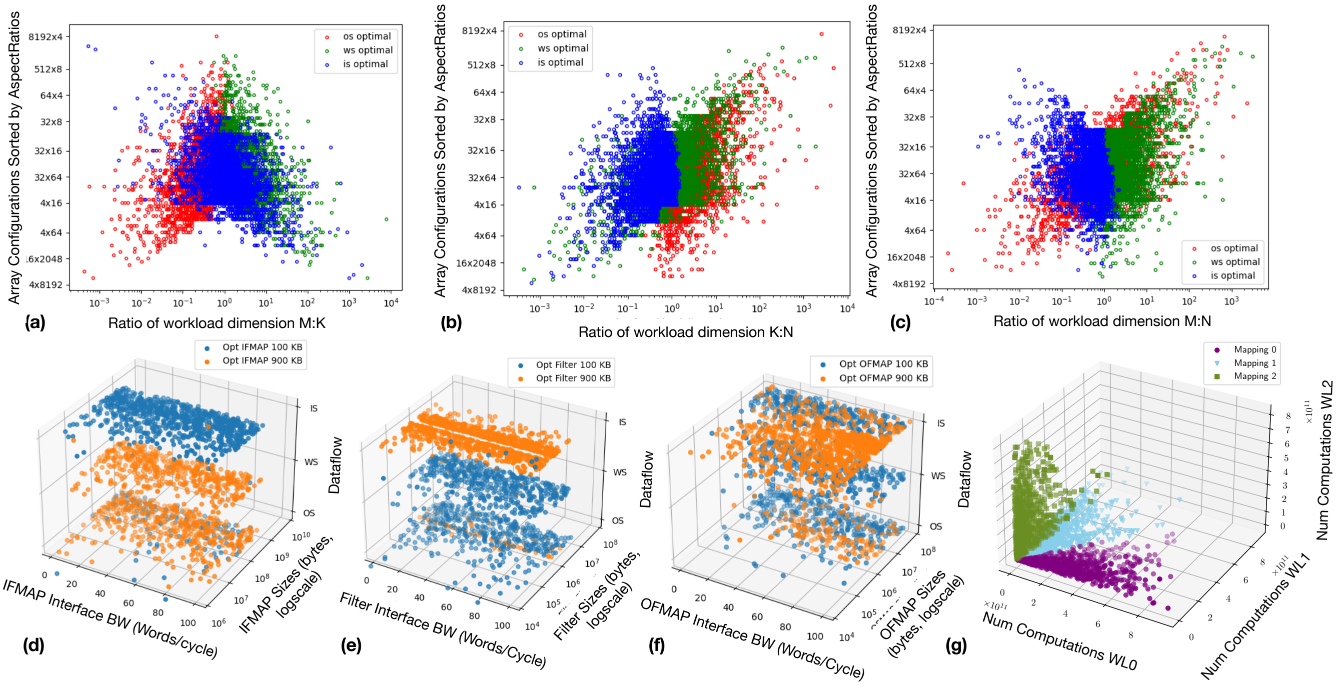

Fig. 6. The correlation of optimal array dimension (y-axis) and the matrix shape as its aspect ratio (x-axis) of (a) Input operand matrix (M×K) (IFMAP), (b) input operand matrix (K×N) (Filter), and (c) Output matrix (M×N) (OFMAP). The marker colors indicate the optimal dataflow, highlighting the pattern in the design space. The relation of optimal buffer sizes of (d) IFMAP operand buffer, (e) Filter operand buffer, and (f) OFMAP buffer, with operand sizes, interface bandwidth, and dataflow. (g) Clusters depicting patterns in schedule recommendation in the space of workload dimensions

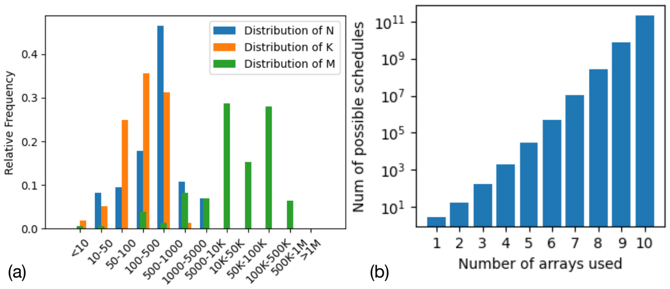

Fig. 7. (a) Chart showing the distribution of operand matrix dimensions for GEMM operations involved in layers of popular neural networks (b) Growth of the scheduling space

units are provided in exponents of 2, limited to a maximum of 2^{18} which is typical for small to large accelerators. *The output space* is a list of labels (Fig. 8(b)), where each label points to a set of parameters, denoting the systolic array height, width, and dataflow. The total number of possible points in this output space is 459. For dataset generation, the optimal parameter label is determined by conventional search using simulations (SCALE-Sim [13]) with runtime as the cost metric.

Case Study 2: Buffer Size prediction. *The input space* comprises 8 integers, which capture the workload dimensions, array dimensions and dataflow, the interface bandwidth of the buffer (bytes/cycle), and the maximum memory capacity(KB) as shown in Fig. 8(a). *The output space* is a set of buffer dimensions, where each buffer size is quantized as multiple of 100 KB with 1MB limit. The size of the output space, as shown in Fig. 8(c), is 1000 points. For each datapoint, the optimal memory size is the one which generates zero or least stalls, while consuming minimum capacity.

	Input Space Dimensions	Parameters
Case Study 1	4	Maximum MAC Units, M, N, K
Case Study 2	8	Buffer Size Limit (KB), M, N, K, Array Rows, Array Cols, Dataflow, Interface Bandwidth
Case Study 3	12	M_{WL0}, N_{WL0}, K_{WL0}, M_{WL1}, N_{WL1}, K_{WL1}, M_{WL2}, N_{WL2} ,K_{WL2}, M_{WL3}, N_{WL3}, K_{WL3}

(a)

Config ID	Array Rows	Array Cols	Dataflow
0	4	4	OS
1	4	4	WS
2	4	4	IS
3	8	4	OS
	⋮	⋮	⋮
458	2^{18}	4	IS

(b)

Config ID	IFMAP Buffer KB	Filter Buffer KB	OFMAP Buffer KB
0	100	100	100
1	100	100	200
2	100	100	300
3	100	100	400
	⋮	⋮	⋮
999	1000	1000	1000

(c)

	Compute Array 0		Compute Array 1		Compute Array 2		Compute Array 3	
Config ID	Workload ID	Dataflow	Workload ID	Dataflow	Workload ID	Dataflow	Workload ID	Dataflow
0	0	OS	1	OS	2	OS	3	OS
1	0	OS	1	OS	2	OS	3	WS
2	0	OS	1	OS	2	OS	3	IS
3	0	OS	1	OS	2	WS	3	OS
	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
1943	3	IS	2	IS	1	IS	0	IS

(d)

Fig. 8. (a) Input space size and input parameters for the three case studies; Output space of (b) systolic array dimension and dataflow prediction case study (c) memory buffer size prediction case study (d) distributed array schedule prediction case study

Case Study 3: Multi-Array Scheduling. *The input space* for this problem, is simply the GEMM workload dimensions (M, N, K) one for each compute array. *The output space*, is an id depicting the mapping of workload to the arrays and the corresponding dataflow to be used as shown in Fig. 8(d). The possible number of scheduling strategies grows exponentially and combinatorially with the arrays count (given by $N = 3^x \times x!$, with x arrays) as shown in Fig. 7(b). *In this case study, we*

SVC RBF	Support Vector Classification using Radial Basis Kernel
SVC Linear	Support Vector Classification using Linear Kernel
XGBoost	eXtreme Gradient Boosting based Classifier
MLP-A	Multi-layer perceptron with 1 hidden layer with 128 nodes
MLP-B	Multi-layer perceptron with 1 hidden layer with 256 nodes
MLP-C	MLP with 2 hidden layer with 128 nodes each
MLP-D	MLP with 2 hidden layer with 256 nodes each
AIrchitect	Custom network shown in Fig1, with 256 hidden nodes

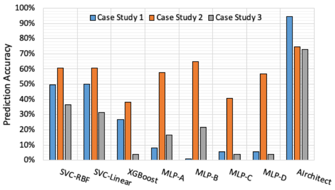

Fig. 9. Performance of classifier frameworks.

chose to learn the scheduling space of ***four*** *arrays, which leads to 1944 possible entries in the output space*. A configuration is deemed optimal if it has the lowest runtime and consumes least energy determined by our in-house simulator.

C. Learning with Existing Classifiers

The problem formulation discussed in Sec. IV allows us to use off-the-shelf classifiers to capture the design space and make predictions about the optimal parameters for given workloads and design constraints. We test out various models with different complexity on the datasets generated for the three case studies. The table in Fig. 9 show these models. We used the scikit-learn libraries implementation of support-vector classifiers with linear and radial basis kernel. The state-of-the-art tree boosting method called eXtrement gradient boosting (XGBoost) available from the xgboost package. We also implemented four multi-layer perceptrons (MLP) in TensorFlow's Keras.

Fig. 9 shows the accuracy obtained for the case studies when 2×10^6 datapoints are used for fit/training. For the MLPs, the networks are trained for 15 epochs with 90:10 training-validation split. We used a categorical cross-entropy loss function with accuracy as the optimization objective. Among the various case studies, the one for memory size prediction is learned well among all the models, with MLP-B attaining about 63% validation accuracy. Support Vector Classification was able to perform the best among the models considered attaining about 50% test accuracy for case study 1, 60% and 40% for the other two, respectively.

V. AIrchitect: Design and Analysis

Recommendation model design. From our analysis in section Sec. IV-C, we notice that although off-the-shelf classifiers are capable of learning the design space, no single model appears to perform consistently across the different case studies. We design a general network structure, which we call AIrchitect, intending to obtain the capability to achieve good learning performance across different distributions. Fig. 2 depicts the general structure of our proposed model. The trained embedding map the input data from the user-defined input space onto a latent space, which immensely improves the performance of the classifiers. This is also evident from the performance of MLP-B vs. AIrchitect as depicted in Fig. 9. Among the various case studies, the number of inputs and outputs are the parameters that we changed. The number of inputs to the network is equal to the input space, while the network generates a one-hot vector of length equal to the output space of the problem indicating the optimal design or mapping parameters, as discussed in Sec. IV-B, Fig. 8. We use an embedding size of 16 and a 256-node MLP hidden layer for our models across the case studies.

Training. For all three use cases we train the corresponding versions of AIrchitect on the respective datasets with 4.5M points (Sec. IV) using 80:10:10 train-validation-test split. We use the Keras library to implement the network and train using categorical-cross-entropy as the loss function, with accuracy as the optimization metric. From Fig. 10(a-c), we observe that the design space of case study 1 is learned with a high validation accuracy ($> 94\%$) in about 15 epochs (1 hour using 1 Nvidia Tesla GPU). The training for case study 2 finishes in about 22 epochs, achieving a validation accuracy of 74% before starting to overfit. For case study 3, the network saturates at about 15 epochs at about 76% validation accuracy. As we see in Fig. 9, AIrchitect beats the best performing off-the-shelf classifiers at least by about 10% accuracy.

Analysis of trained model. We plot the distribution of the actual labels and the predicted configuration IDs for 10^5 previously unseen test data points for each case study. Fig. 10(d-f) shows predicted vs actual distribution for the three case studies. *First*, we notice, visually the predicted distribution for case study 1 almost perfectly matches the actual distribution, confirming that the network learned the design space. *Second*, the networks for case studies 2 and 3 learn to predict the configurations with significant representation on the dataset, while the configuration IDs with low statistical probability are ignored as noise. The large spikes in the memory size prediction induce a bias to the model. This bias is shown by the high frequencies of the top two configuration IDs (Fig. 10(e)), and it leads to relatively low accuracy. Similarly, for case study 3, Fig. 10(f) shows that the model was successful in learning the distribution of the large spikes. However, it ignored the configurations with lower frequencies, which in turn cumulatively lowered the accuracy figure. *It is worth noting that the model ignores the lower frequencies as statistical noise; this demonstrates that it is robust and does not overfit for any of the use cases.*

Misprediction penalty. In Fig. 10(g), we show the performance of the predicted configurations normalized to the labels for the 10^5 datapoints. We observe that only a few data points have catastrophic performance losses (<20% of the optimal), leading to a 99.99% average performance (Geometric Mean) of the best possible. Interestingly, for case study 3, which had relatively low accuracy, the performance does not lead to catastrophic losses for most cases. Fig. 10(h) shows that among the mispredictions, most of the points suffer about 10% to 15% loss compared to the best achievable, leading to an average (GeoMean) of 99.1% of the best runtime.

Performance on existing network layers. In Fig. 11(a) we depict the performance of AIrchitect for case study 1 on a few layers from networks like FasterRCNN, GoogleNet, Alexnet, MobileNet, and ResNet-18. No layer of these networks were part of the training or validation dataset, but the model can predict the optimal array shapes and dataflow when queried with a budget of 2^{10} MAC units.

Performance at scale. Fig. 11(b) we show the performance of the recommender when trained on variations of datasets for case study 1, with increasing MAC unit budget. A larger budget leads to increase in number of possible configurations

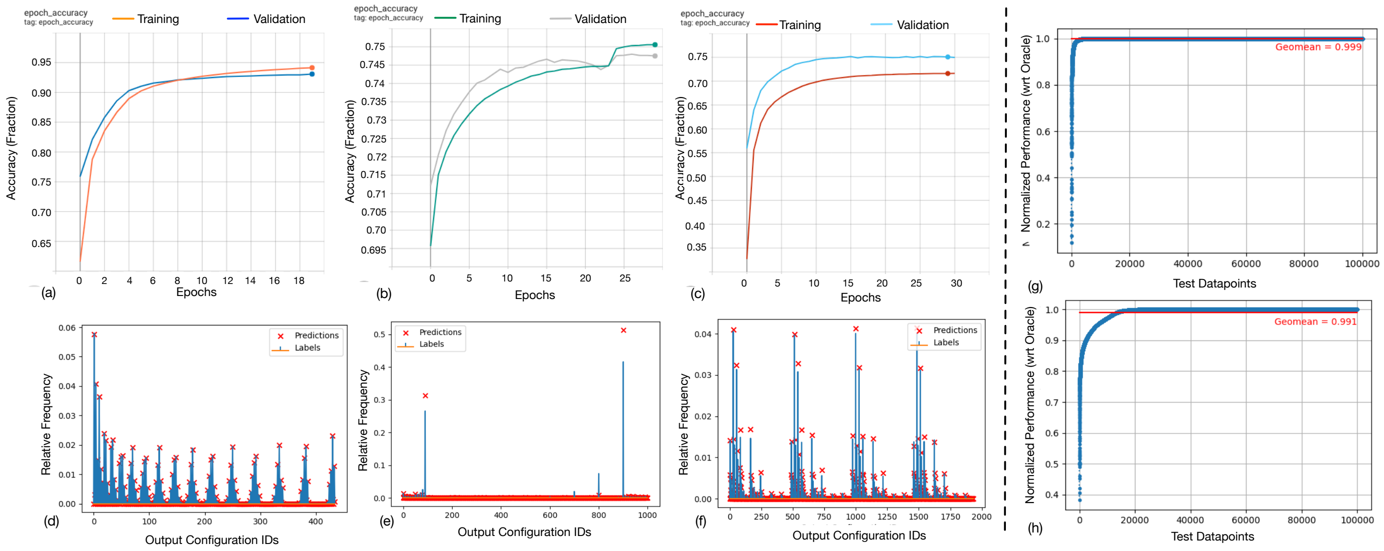

Fig. 10. Progression of training and validation accuracy vs. epochs when training AIRCHITECT on the dataset for (a) Case Study 1, (b) Case Study 2, (c) Case Study 3. The distribution of actual label and predicted configuration IDs by trained AIRCHITECT on test datasets with 10^5 points for (d) Case Study 1, (e) Case Study 2, (f) Case Study 3. The sorted normalized performance of predicted configurations to the optimal configurations on workloads present in the test dataset for (g) Case Study 1, (h) Case Study 3

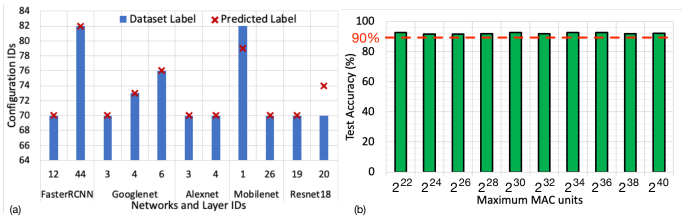

Fig. 11. Predicted and actual labels for case study 1 for a few layers of popular CNNs

and thus the size of the output space. We observe that the network consistently provides test accuracies of >90% even with large design spaces with 2^{40} MAC units.

VI. RELATED WORKS

ML for Architecture search: Apollo [15] targets efficient sample searching through the accelerator design space using RL. Gamma [8] and ConfuciuX [7] are similar ML-based architecture mapping and design-space-configuration search methods that use a GA and RL, respectively. AutoTVM [2] uses an ML model for cost prediction to improve fast mapping search at compile time.

ML for EDA: Recently there has been a significant push toward automating place-and-route tools using ML. NVCell is an RL-based proposal from Nvidia to automate standard cell placement. Nautilus uses a GA to improve FPGA place and route. Kwon et al. [10] use online tensor-based recommender systems to aid place and route in chip design.

VII. CONCLUSION

This paper presents AIRCHITECT, a network design and methodology for constant time architecture design and mapping optimization. Compared to the contemporary method of using ML for speeding up search based optimization, we propose using a learn model to predict the optimal architecture and mapping parameters. The paper systematically analyses the design space, describes the methodology and evaluates the prediction performance of the proposed model. We plan to extent this methodolgy to other design spaces in the future.

VIII. ACKNOWLEDGEMENT

We thank Matthew Denton for his help. We also thank the reviewers for their suggestions to improve the paper.

REFERENCES

[1] O. Azizi *et al.*, "Energy-performance tradeoffs in processor architecture and circuit design: a marginal cost analysis," *ACM SIGARCH Computer Architecture News*, 2010.

[2] T. Chen *et al.*, "Learning to optimize tensor programs," in *NeurIPS*, 2018.

[3] Y.-H. Chen *et al.*, "Eyeriss: A spatial architecture for energy-efficient dataflow for convolutional neural networks," *ISCA*, 2016.

[4] C. Dubach *et al.*, "Microarchitectural design space exploration using an architecture-centric approach," in *MICRO*, 2007.

[5] E. İpek *et al.*, "Efficiently exploring architectural design spaces via predictive modeling," *ACM SIGOPS Operating Systems Review*, 2006.

[6] N. P. Jouppi *et al.*, "In-datacenter performance analysis of a tensor processing unit," in *ISCA*, 2017.

[7] S.-C. Kao *et al.*, "Confuciux: Autonomous hardware resource assignment for dnn accelerators using reinforcement learning," in *MICRO*, 2020.

[8] S.-C. Kao and T. Krishna, "GAMMA: automating the HW mapping of DNN models on accelerators via genetic algorithm," in *ICCAD*, 2020.

[9] S. Khan *et al.*, "Using predictivemodeling for cross-program design space exploration in multicore systems," in *PACT*, 2007.

[10] J. Kwon *et al.*, "A learning-based recommender system for autotuning design fiows of industrial high-performance processors," in *DAC*, 2019.

[11] B. C. Lee and D. M. Brooks, "Accurate and efficient regression modeling for microarchitectural performance and power prediction," *ACM SIGOPS operating systems review*, 2006.

[12] L. Mei *et al.*, "Zigzag: Enlarging joint architecture-mapping design space exploration for dnn accelerators," *IEEE Computers*, 2021.

[13] A. Samajdar *et al.*, "Scale-sim: Systolic CNN Accelerator Simulator," *arXiv preprint arXiv:1811.02883*, 2018.

[14] ——, "A Systematic Methodology for Characterizing Scalability of DNN Accelerators using Scale-Sim," in *ISPASS*, 2020.

[15] A. Yazdanbakhsh *et al.*, "Apollo: Transferable architecture exploration," *arXiv preprint arXiv:2102.01723*, 2021.