MODELING DNN LAYER PERFORMANCE ACROSS ACCELERATOR DESIGNS

CS 267 FINAL PROJECT

Charles Hong
University of California, Berkeley
charleshong@berkeley.edu

November 5, 2021

ABSTRACT

In recent years, the popularity of deep neural networks (DNNs) has led to the development of specialized DNN accelerators that utilize large numbers of parallel processing elements to improve throughput. Like other computational hardware, DNN accelerators must schedule their workloads in order to optimally parallelize a DNN layer on the given hardware. However, the configuration of the accelerator can have a large impact on what schedules are optimal, or even valid. In this project, we evaluate three main machine learning models on the task of predicting deep neural network layer runtime on new, unseen parallel accelerator designs. We present an analytically preprocessed feature set that improves the performance of our random forest model beyond the other two, and show the usefulness of MDI feature importance analysis with random forest models, utilizing the method to identify the most predictive input features. Finally, we show that the random forest model can accurately predict layer runtime on unseen accelerator designs as well as predict whether schedules are valid or invalid.

Keywords Parallelism · Deep neural networks · Computer architecture · Accelerators

1 Introduction

Specialized deep neural network (DNN) accelerators, such as Google's Tensor Processing Unit (TPU), are becoming more and more common. In order to use these accelerators, compilers must schedule nested loop workloads, such as convolutional neural network layers, by defining their ordering, tiling, and spatial mapping (which dimensions are parallelized). Currently, these schedules often do not optimally parallelize their workloads [1].

By accurately modeling the performance and energy usage across different DNN layers and accelerator architectures, we can find DNN layer schedules that more optimally parallelize computation, even on hardware configurations that have not yet been deployed. This would allow for rapid software-hardware co-design of machine learning compilers and DNN accelerators, bypassing long simulation times that are often a barrier to optimal hardware design [2]. In addition, such a model could allow for co-scheduling of DNN layers on a partitioned accelerator, which would provide an additional level of parallelism to spatial accelerators like the TPU, which are already inherently parallel. The goal of this work is to understand whether it is possible to build this type of generalizable model, while keeping it accurate and interpretable enough to inform future design choices.

2 Background

This project builds on CoSA (Constrained Optimization for Spatial Accelerators) [1], which focuses on identifying an optimization strategy for DNN layer scheduling on the Simba accelerator [3]. CoSA uses mixed integer programming techniques to solve for the optimal performance and energy given resource constraints. To evaluate a schedule, CoSA integrates Timeloop [4], which provides microarchitecture- and technology-specific energy models that help estimate performance and energy of the accelerator. Additionally, there are other works related specifically to modeling DNN

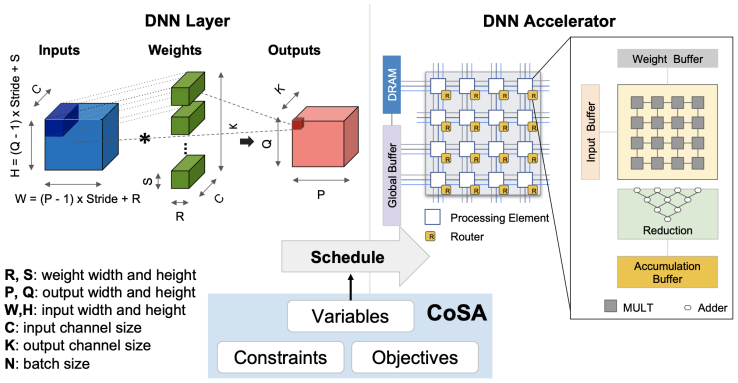

Figure 1: From "CoSA: Scheduling by Constrained Optimization for Spatial Accelerators" [1]

accelerator performance. Figure 1 shows where CoSA fits in to the DNN flow. Particularly relevant to this work are the constraints in CoSA's schedule optimization: buffer capacity constraints (whether the schedule calls for an amount of on-chip buffer usage that exceeds the amount available) and spatial constraints (whether the schedule calls for parallel computation that exceeds the number of parallel compute units available).

3 Related Work

3.1 A Learned Performance Model for the Tensor Processing Unit

In this work, Kaufman et al. [5] train a neural network over kernel-level sub-graphs of a neural network to predict its performance on a TPU. This allows for optimization of the operator fusion configuration and layer tile-size selection. This work focuses on building an end-to-end performance model for a neural network, targeted specifically to the TPU. The performance model is then used to tune ML compilers for this specific accelerator. In contrast, our performance model seeks to predict parallel performance on arbitrary accelerator configurations, with the goal of enabling faster hardware-software codesign.

3.2 Mind Mappings

Another prior work that addresses a similar problem is Mind Mappings [6]. This work focuses on choosing DNN schedules based on fast, gradient-based search algorithms. The authors enable the use of such algorithms by collecting data on the performance and energy usage of broadly sampled DNN layer schedules, and using it to train a surrogate multi-layer perceptron (MLP) model. Mind Mappings motivates our work by indicating that a surrogate machine-learning based model for neural network performance prediction can provide enough accuracy to inform neural network scheduling on accelerators. In this project, we extend this idea to both neural network scheduling and accelerator design, and present new insights on what types of models can be used. Like the previous work, Mind Mappings trains a performance model for a specific hardware configuration.

4 Infrastructure

A significant portion of this project's contribution is in building infrastructure that allows the fast evaluation of different machine learning models on different accelerator designs and neural network layer schedules. This is because in order to evaluate the performance and usefulness of different types of models, we first need to construct our dataset. This task took place in two parts.

4.1 Accelerator Generation

In order to model DNN layer performance on different hardware, we generate a wide range of accelerator descriptions that could be passed to Timeloop. The amount of parallel processing hardware is defined by the number of multiply-accumulate units (MACs), which are divided among the processing elements (PEs). Each PE's on-chip memory is divided amongst the accumulation, weight, input, and global buffers, which can each have different problem data mapped into them. For some experiments, a limited number of accelerator configurations was used, including `simba_final`,

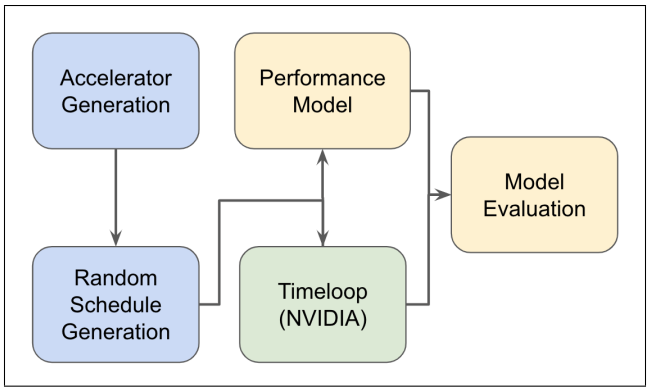

Figure 2: Summary of the approach taken.

Table 1: Preset accelerator parameters

Parameter	`simba_final`	`simba_final_8x8`	`simba_large`	`arch0`	`arch1`
No. of PEs	16	64	16	16	16
No. of MACs	1024	4096	1024	1024	4096
Accumulation buffer size / PE	3 KiB	12 KiB	6 KiB	6 KiB	3 KiB
Weight buffer size / PE	32 KiB	128 KiB	64 KiB	64 KiB	16 KiB
Input buffer size / PE	8 KiB	32 KiB	16 KiB	16 KiB	16 KiB
Global buffer size / PE	64 KiB	256 KiB	128 KiB	32 KiB	64 KiB

Table 2: Generated accelerator parameters (2304 configurations)

Parameter	Minimum value used	Maximum value used
No. of PEs	4	64
No. of MACs	256	4096
Accumulation buffer size / PE	1.5 KiB	12 KiB
Weight buffer size / PE	8 KiB	128 KiB
Input buffer size / PE	2 KiB	32 KiB
Global buffer size / PE	16 KiB	256 KiB

the accelerator configuration used by Simba [3]. These configurations can be found in Table 1. For other experiments, we generated 2304 different accelerator descriptions with parameters throughout the ranges given in Table 2.

4.2 Schedule Generation

Another challenging aspect of data generation is generating schedules throughout the search space for each accelerator configuration. In order to randomly generate schedules, we consider the loop nest representation in Figure 3. A deep neural network layer, for example a convolutional layer, can be thought of as a program that iterates through various problem dimensions, including the input and output width and height, weight width and height, input and output channel sizes, and batch size. The size of each dimension can be decomposed into its prime factors. These can be used to determine the following:

- **Loop permutation**, the order of the dimensions in the computational loop nest can be permuted depending on the specific problem dimensions.
- **Memory mapping**, how much of each problem dimension should be mapped to each memory level. This determines buffer usage.
- **Spatial mapping**, how much parallel computation is allotted to a dimension. This determines parallel spatial resource utilization.

A schedule can either be valid, meaning it does not exceed the hardware resources available, or invalid, meaning computational or memory requirements are not met. For each of the 2304 generated accelerator configurations, we randomly set the loop permutation, loop tiling, and spatial mapping of each dimension using its prime factors until

```
//DRAM level
for q2 = [0 : 2) :
 // Global Buffer level
 for p2 = [0 : 7) :
  for q1 = [0 : 7) :
   for n0 = [0 : 3) :
    spatial_for r0 = [0 : 3) :
     spatial_for for k1 = [0 : 2) :
      // Input Buffer level
      spatial_for k0 = [0 : 2) :
       // Weight Buffer level
       for c1 = [0 : 2) :
        for p1 = [0 : 2) :
         // Accumulation Buffer level
         for s0 = [0 : 3) :
          for p0 = [0 : 2) :
           spatial_for c0 = [0 : 8) :
            // Register
            for q0 = [0 : 2) :
```

Figure 3: An example loop nest representation of a DNN layer schedule. (From "CoSA: Scheduling by Constrained Optimization for Spatial Accelerators" [1])

we had 10 valid and 100 invalid schedules for each of the 24 computationally distinct layers of ResNet-50, a popular convolutional neural network used for computer vision. This resulted in a total of around 6 million schedules, around 9 percent of them valid. We also used a separate dataset of 3000 valid schedules per layer for each of the 5 preset accelerator configs in Table 1, for a total of 360,000 schedules.

5 Modeling

5.1 Features

All combined, the accelerator and schedule parameters in Section 4 result in a total of 132 input features, including 126 schedule features. In order to reduce the computational cost of training our model, we experiment with the use of prior work from CoSA [1] to analytically combine the 126 schedules features into 30 features that encode a similar amount of information. This is done through the following methods:

- Estimate the utilization of each buffer by each data type (weights, input activations, and output activations) based on memory mappings.
- For each buffer, multiply the spatial mapping factors of each problem dimension that are mapped to that buffer.
- Estimate global buffer and DRAM communication cost using the traffic-driven objective from CoSA [1].

These new input features can aid in prediction by adding doing some processing ahead of time. The compressed feature set also significantly reduces training time—for example, our random forest model takes 21 minutes to train on around 6 million schedules with 132 input features, and just 11 minutes with 36 input features.

5.2 Models

Table 3: Models used

Model type	Description	Interpretability	Training time
Linear regression	Ordinary least squares	High	Low
Random forest	1000 trees, no max depth	Medium	High
Multi-layer perceptron	Hidden layer dimensions: (64, 32, 16, 4)	Low	High

This project focuses on the evaluation of three basic types of machine learning models: linear regression, random forest, and multi-layer perceptron. A description specific models used can be found in Table 3, along with a general classification of each model's interpretability and training time. In recent years, neural network models like multi-layer perceptrons have become highly popular modeling tools due to their accuracy and differentiability, as in Mind Mappings [6]. However, a significant drawback of neural network models is their lack of interpretability. Though many efforts

have been made to understand the meaning of neural network parameter values, this is still a largely unsolved problem [7]. On the other hand, linear regression coefficients provide direct insight into how each input feature affects the output.

Random forests lie somewhere in between. Because random forests predict by averaging the outputs of many decision trees, it is not immediately obvious how each input feature affects the regression output. However, some insight can fairly easily be gained through a method called feature importance, specifically through Mean Decrease Impurity (MDI). In MDI, a given feature's importance is calculated by summing up the decrease in Gini impurity associated with all splits performed according to that feature [9]. The individual trees in a random forest can also be visualized, though this does not fully express how the random forest predicts.

6 Experiments

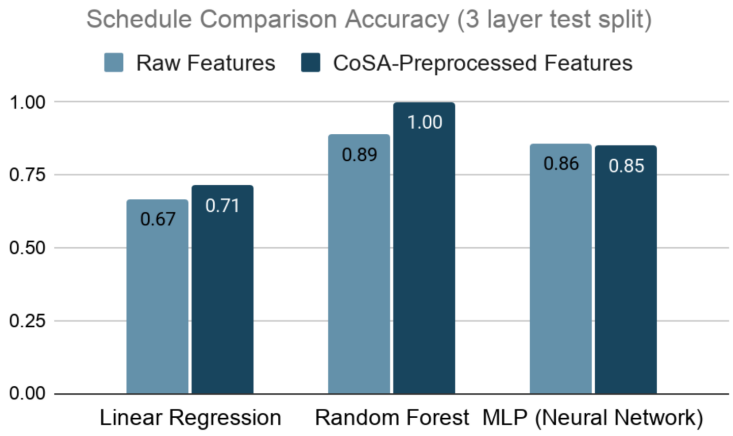

Figure 4: Schedule comparison accuracy for the three model types, trained on 3000 schedules/layer on 5 different accelerators.

Figure 6 summarizes the generalizability of each model in Section 5.2 to new, unseen layer schedules. Each model is trained to predict the cycle count of a schedule on 3000 schedules per layer of the 5 accelerators configurations from Table 1, on 21 of the 24 computationally unique layers of ResNet-50, with the original set of 132 accelerator and schedule parameters as input features. Schedules for three remaining layers make up the test set; two of the test set layers are convolutional layers, and one is a fully connected layer. For each accelerator, for each of these three layers, the model is evaluated on all pairs of schedules. The "Schedule Comparison Accuracy" of a model is the proportion of pairs where the model accurately predicts which schedule is slower (has a higher cycle count), and which is faster (has a lower cycle count).

Unsurprisingly, linear regression performs the worst at 67% accuracy. Using the 132 original input features, the random forest and MLP models perform similarly, with 89% and 86% accuracy respectively.

6.1 CoSA-Preprocessed Features

Also in Figure 6 are the results of the same experiment, using instead the compressed set of 36 analytically preprocessed features. With this new feature set, the linear regression and MLP models maintain similar accuracy as with 132 features. The random forest model, however, sees its accuracy increase to nearly 100%. We hypothesize that this is because the preprocessed feature set encodes most of the same information as before while significantly reducing dimensionality, meaning it encodes more information per feature. This increases the predictive power of each split in the decision trees that make up the random forest, greatly improving the random forest's efficiency.

This explanation is supported by the MDI feature importance analysis of the random forest models trained on the two sets of features. With the original feature set, there are several features with an MDI of 0.06–0.1. With the preprocessed feature set, the most important feature, `utilized_spatial_factors_memlvl4`, has an MDI of over 0.5. This feature is the product over all problem dimensions of the spatially mapped prime factors that are allocated to the global buffer of the accelerator. It makes sense that this value is highly predictive of DNN layer schedule runtimes, since memory access can be a significant portion of a program's runtime, and the global buffer is the largest of the local buffers.

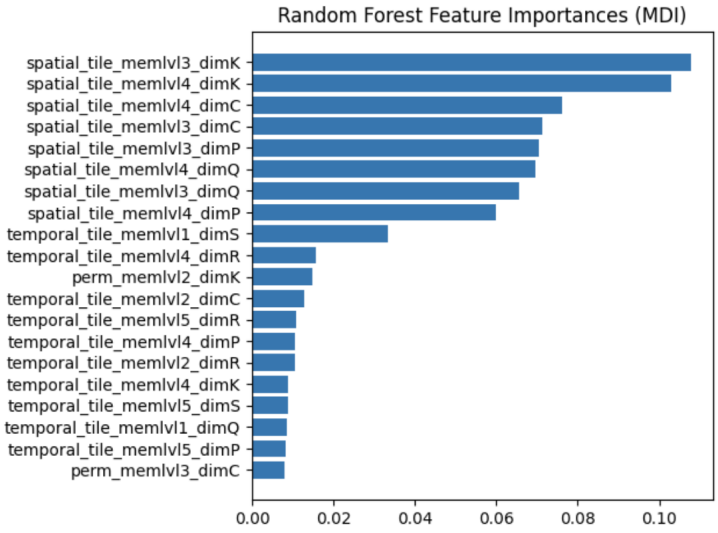

Figure 5: Random forest feature importance of the original 132 input features, using the MDI method.

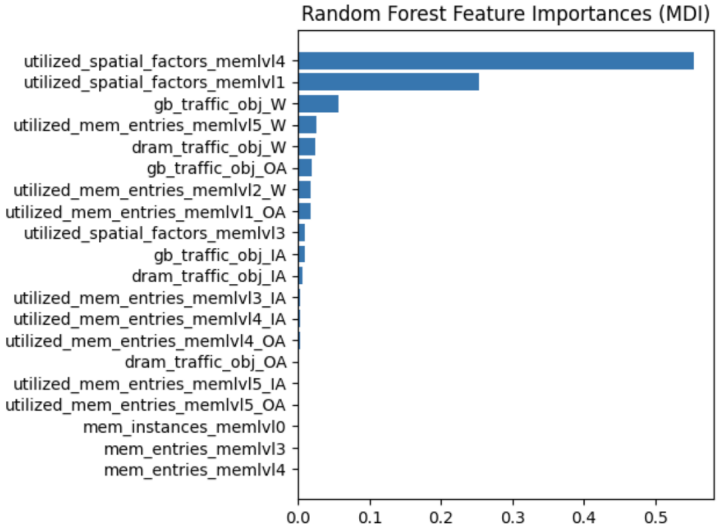

Figure 6: Random forest feature importance of the preprocessed set of 36 input features, using the MDI method.

One investigation we are not able to fully make in this work is into the meaning of certain schedule features with respect to accelerator design. For example, it is possible that the importance of global buffer tiling factors means it should be smaller in order to reach max efficiency, because other buffers may be underutilized relative to the global buffer. However, it is also possible that the importance of global buffer tiling factors indicates that increasing these factors has an outsized positive effect on performance, meaning the buffer should be larger. We would like to better explore these effects in future work.

We also found that the random forest was far able to far more accurately predict runtimes of the last (fully connected) layer of ResNet-50 than the MLP, with schedule comparison accuracies of 97% and 50% respectively. This may be because a fully connected layer has very different problem dimensions from a convolutional layer, with much larger weights and only one input and output channel. While the neural network treats such a layer as an outlier, it is possible that a random forest contains trees that have branches specifically for fully connected layers. In future work, this hypothesis could be further tested by inspecting the trees in the random forest.

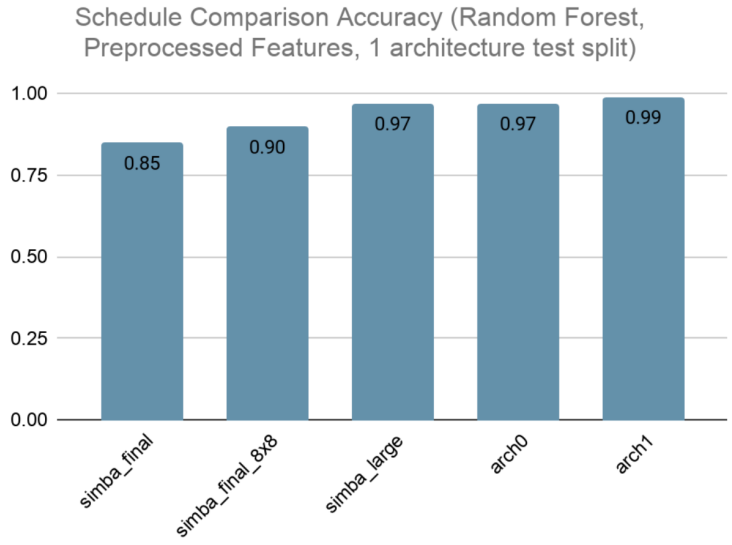

Figure 7: Generalizability of a random forest model to new accelerator architectures.

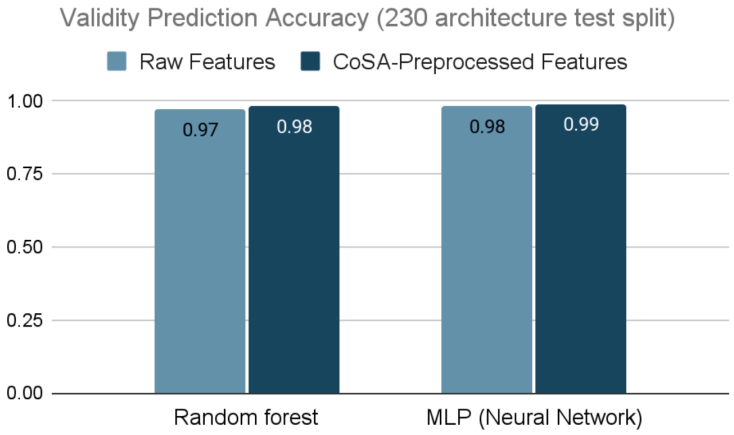

Figure 8: Schedule validity prediction accuracy, tested on 10% of 2304 different accelerators.

6.2 Model Generalizability to New Architectures

Figure 7 shows the accuracy with which the random forest model compares schedules for a given accelerator, when trained on the four other accelerators in the smaller dataset. The results, though fairly limited in scope, range from

85% to 99% in accuracy, showing that a random forest is able to new, unseen architectures in some cases. One reason why this may be the case is that our analytically preprocessed features inherently encode some information about architectural parameters. For example, the amount of utilized memory entries relates to the size of a buffer. If an accelerator's buffer size is increased, then in order for a layer's runtime to change, the number of utilized memory entries should increase to take advantage of it. However, the effect of the change memory utilization on the layer's runtime may already by encoded in the machine learning model. Additionally, if a memory buffer size is increased, but the amount of utilized memory entries does not change, then the runtime of such a schedule should remain constant. In future work, we would like to investigate whether our models are able to identify the roofline of an accelerator design, and understand whether performance is being limited by computational or communication resources [10].

6.3 Schedule Validity

We also consider whether machine learning models can be used to predict if a schedule is valid or not. This study uses the dataset of 2304 accelerator architectures, with 10 valid and 100 invalid schedules per layer. The test data consists of schedules for 230 of these accelerators, which are entirely omitted from the training data. Shown in Figure 8, both the random forest and MLP models have a near-100% accuracy on this 0/1 classification task, showing that avoiding invalid schedules should not be a problem if these models are used to generate novel schedules in future work.

7 Conclusion and Future Work

In this project, we evaluate three main machine learning models on the task of predicting deep neural network layer runtime on new, unseen parallel accelerator designs. After a thorough data analysis, we find that surprisingly, when combined with a compressed, analytically preprocessed feature set, the most accurate of the three models is the random forest. We also show the usefulness of MDI feature importance analysis with random forest models, utilizing the method to identify the most predictive input features. Finally, we show that a random forest model has high potential to predict layer runtime on unseen accelerator designs, and that statistical methods can be used to avoid invalid schedules.

There are many potential avenues for extending this project. The analysis presented is meant to serve as a building block towards future exploration of hardware-software co-optimization for deep neural network accelerators. Further analysis of the models themselves could provide new insights into the importance of specific accelerator or schedule parameters. For example, as noted in Section 6.1, the MDI feature importance of specific schedule attributes are also likely related related to the hardware they are run on, meaning that they schedule data may actually provide insight on the effectiveness of a hardware design. With further tuning, the models could be part of generative or reinforcement learning-based methods for generating new accelerator designs or more optimally parallel schedules.

8 Acknowledgements

Many thanks to Jenny Huang for help with CoSA and building the project infrastructure, Professor Sophia Shao for guidance, and the CS 267 course staff (particularly Professor James Demmel) for feedback.

References

[1] Huang, Q. et al. “CoSA: Scheduling by Constrained Optimization for Spatial Accelerators.” *International Symposium on Computer Architecture (ISCA)*, 2021.

[2] Yazdanbakhsh, A. et al. "Apollo: Transferable Architecture Exploration" *arXiv preprint*, 2021.

[3] Shao, Y.S. et al. “Simba: Scaling Deep-Learning Inference with Multi-Chip-Module-Based Architecture.” *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture*, 2019.

[4] Parashar, A. et al. "Timeloop: A Systematic Approach to DNN Accelerator Evaluation." *IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, Madison, WI, USA, 2019, pp. 304-315, doi: 10.1109/ISPASS.2019.00042.

[5] Kaufman, S.J., Mangpo Phothilimthana, P., Zhou, Y., and Burrows, M. “A Learned Performance Model for the Tensor Processing Unit.” *Proceedings of the 4th MLSys Conference*, 2021.

[6] Hegde, K. et al. “Mind Mappings: Enabling Efficient Algorithm-Accelerator Mapping Space Search.” *International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2021.

[7] Samek, W. et al. “Explainable artificial intelligence: Understanding, visualizing and interpreting deep learning models.” *arXiv preprint*. arXiv:1708.08296, 2017.

[8] Wojtas, A. et al. “Feature Importance Ranking for Deep Learning.” *arXiv preprint*, 2020.

[9] Scornet, E. "Trees, forests, and impurity-based variable importance." *HAL preprint*, 2020.

[10] Williams, S. et al. "Roofline: An Insightful Visual Performance Model for Floating-Point Programs and Multicore Architectures." *Communications of the ACM*, 2009.