Transformers in Reinforcement Learning for Hardware Design Space Exploration

Junsun Choi *Charles Hong*
Electrical Engineering and Computer Science, University of California, Berkeley

Abstract

As domain-specific accelerators have become more commonplace, efficient design space exploration (DSE) in hardware designs has become a crucial problem. The problem in hardware design space exploration is that it takes a significant amount of time per one design iteration. To deal with this problem, a DSE algorithm should find a near-optimal hardware design in as few invocations of the expensive hardware simulator as possible. In cases where the algorithm uses a performance model in order to do so, the model should be as accurate as possible. We propose a novel approach where we replace the RNN policy network with a transformer in the reinforcement learning (RL)-based deep learning accelerator optimizer, ConfuciuX. We also perform a thorough hyperparameter search to identify optimal model parameters. We show that our solution, ConfuciuX-TF, improves searched hardware performance by 12.1%, while reducing runtime by 27% compared to the original solution.

1 Introduction

With the end of Moore's Law and Dennard Scaling, the performance improvement of general purpose processors has slowed down. Nowadays, the general consensus is that developing Domain-Specific Accelerators (DSAs) is necessary to continue the rapid trend of pre-Moore performance gains. However, the rise of DSAs has resulted in increased non-recurring engineering (NRE) cost to develop hardware. Therefore, efficient design space exploration (DSE) in hardware designs is a crucial problem.

The problem in hardware design space exploration is that it takes a significant amount of time to run a simulator to test one hardware design configuration, and even more time to do post-silicon verification. However, modifications in hardware architectural design can yield many-fold improvements under target metrics such as power, performance, and chip area (PPA). Considering the fact that the time given for product launch is usually limited, it is infeasible to exhaustively search the design space of hardware configurations. And while human experts can attempt the use of various heuristics to find an optimal design point, there is no guarantee that it is the optimal point. In fact, prior work has shown the effectiveness of machine learning algorithms in improving the efficacy of hardware architectural design choices versus human choices [11].

To deal with the problem mentioned above, a DSE algorithm should find a near-optimal hardware design in as few invocations of the expensive hardware simulator as possible. In case where the algorithm uses a performance model in order to do so, the model should be as accurate as possible. Various machine learning models have been applied to this end, such as variational autoencoders [7]. Another system, ConfuciuX [13], uses reinforcement learning to optimize deep learning accelerator parameters for a set of layers. We propose ConfuciuX-TF, a modified solution that seeks to improve both agent performance and runtime.

The ConfuciuX-TF source code can be found on GitHub at `https://github.com/charleshong3/cs285-proj`.

2 Background and Motivation

2.1 Deep Learning Accelerators

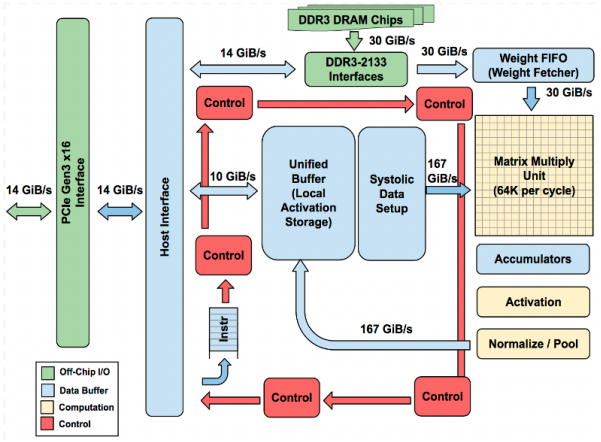


Figure 1: TPU v1 block diagram, from [12]. As described in 2.1, the yellow grid represent the 2D array of multiply-accumulate units, and the blue blocks represent the on-chip buffers that store data locally for faster access.

As previously mentioned, domain specific accelerators, particularly those targeted towards deep learning workloads, have become highly popular. The most well-known among these deep learning accelerators is Google's Tensor Processing Unit (TPU), which John Hennessy and David Patterson described as ushering a renaissance of computer architecture in their 2017 Turing Award lecture [6]. Since cost-, area-, and power-efficiency are the main advantages of hardware acceleration, these accelerators must be carefully designed to achieve optimality on one or more of these metrics. The basic deep learning accelerator design, which has not deviated significantly from that of the TPU v1 and its contemporaries, consists of a number of multiply-accumulate units laid out in a 2D array (as in the TPU [12]) or in a SIMD vector layout (as in NVIDIA's NVDLA), along with static random-access memory (SRAM) units. The multiply-accumulate units compute hundreds, or even thousands, of the arithmetic operations that make up a matrix multiplication in one cycle. The SRAMs serve as buffers that locally store the weight, input, and output tensors which would otherwise have to be fetched from DRAM in a high-latency operation, allowing the plentiful arithmetic units to be utilized a higher percentage of the time. Modern neural networks can reach into the billions of parameters in size, as shown in Figure 2, so these SRAMs cannot possibly be made large enough to store all data needed for even one layer at a time. The sizes of the matrix multiplication unit and local buffers must be tuned carefully to achieve optimal efficiency on one of the aforementioned metrics. In addition to the size of each unit, deep learning accelerators also often have parameters such as dataflow, SRAM banking and ports, on-chip network architecture, and data type support that might change from generation to generation. For this project, we focus on matrix multiplication unit and buffer sizing as these parameters are the focus of ConfuciuX.

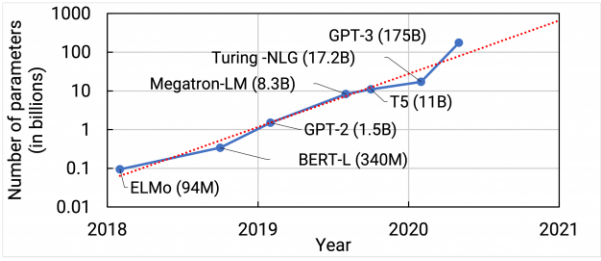


Figure 2: Exponential growth of state-of-the-art AI models. Figure from [4].

2.2 Hardware Simulation Cost

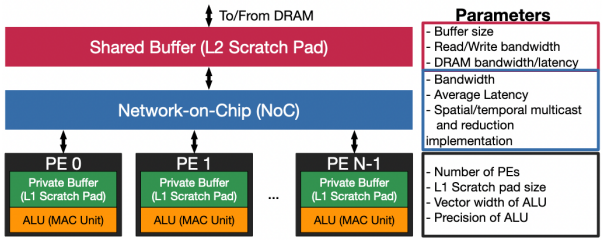


Figure 3: High-level architecture of an accelerator as represented within an analytical performance model, MAESTRO. The figure also shows the variety of parameters that can be reconfigured and make up a large, challenging search space. Figure from [15].

One of the main tasks of computer architects is to optimize important parameters of a hardware design. However, evaluating hardware performance

tends to be costly. Since producing a real chip takes months, architects typically evaluate candidate architectures on a range of other platforms. These platforms include register transfer-level (RTL) simulation using electronic design automation (EDA) tools such as Synopsys VCS or the open-source alternative Verilator, but these tools are typically slow (on the order of magnitude of hours for a single evaluation), since they must reproduce the switching behavior of the entire design over the course of an entire software program. A faster alternative is FPGA-based simulation using tools such as FireSim [14], which can run at hundreds of MHz and complete an evaluation in minutes, but these require each unique hardware design to be compiled into an FPGA-friendy binary format, which can take tens of minutes. Emulation platforms such as Cadence Palladium work similarly and run even faster, but are not available for the typical user due to their high cost.

The fastest evaluation method is by analytical model, for example Timeloop [18] or MAESTRO [15], which are both analytical models targeted towards deep learning accelerators. These tools can run an evaluation in less than a second, but are far less accurate than the previously described methods. Rather than reproduce the behavior of hardware cycle by cycle, they generate an estimate of accelerator performance by calculating the latency of high-level operations such as memory reads and writes and matrix multiplications, and estimating the amount of time-domain overlap between such operations. In our work, we use MAESTRO as our evaluation platform, as this was the platform used for ConfuciuX, and allows us to evaluate our methods in a reasonable amount of time. However, industry teams typically do not rely on such coarse-grained models to make important design decisions that will affect the manufacturing of thousands or millions of chips. Therefore, it is important to develop algorithms that will reduce the number of expensive simulations that are required to identify the optimal parameterization for a given hardware design.

Search Method	Works
Heuristics-Driven:	Interstellar [21]
Black-Box Optimization:	Bayesian Optimization [20] Apollo: P3BO [22] NAAS: Evolutionary [17]
Gradient-Based Optimization:	EDD [16] DiffTune [19]
Search Space Optimization:	VAESA [7]
Reinforcement Learning:	ConfuciuX [13] Jiang et al. [9, 10] ConfuciuX-TF (This Work)

Table 1: Accelerator design space exploration approaches. Due to the popularity of machine learning and deep learning in recent years, many data-driven approaches have been implemented to attempt to reduce hardware search time.

3 Related Work

Despite several years of iteration on the TPU and similar architectures, it is still not possible to achieve 100% utilization of hardware resources. However, there has been a significant amount of research effort and progress towards this end. In particular, researchers have applied ideas from the fields of optimization and machine learning to lessen the cost of searching for the best hardware design point. In this section, we describe previous works in this domain.

3.1 Hardware Design Space Exploration

Architects have been taking a data-driven approach to hardware design space exploration for several decades. In particular, as multiprocessor CPUs became more popular in the early 2000s, making CPU architectures significantly more complicated, the For example, Ipek at al. [8] used neural networks as early as 2006 to predict performance given CPU and workload parameters. With the extreme popularity in recent years of machine learning in almost every application, various data-driven approaches have been attempted. We attempt to characterize the research area in Table 1, though there are many other works that are not cited.

3.2 Heuristics-Driven Approaches

One historically common approach to DSE is where hardware designers use known heuristics to prune the search space, then use random or brute-force search to explore the remaining space. This approach is efficient if the pruned space is well-defined, but it is possible that optimal search points may be eliminated if pruning is done incorrectly. The method is also impossible if heuristics are not available for a certain problem. An example of this approach is Interstellar [21], in which the accelerator search space is limited to sets of dataflows and memory buffer ratios that are known to perform well.

3.3 Black-box Optimization

Another common approach to design space exploration is black-box optimization, which typically focuses on optimizing a task with an unknown objective function but which can return a reward function to provide feedback to the optimizer. Perhaps the most common among these algorithms is Bayesian optimization, which maintains a statistical model of an unknown objective function. In Bayesian optimization, an acquisition function is applied to ensure that a wide range of points are explored, while still exploiting known good regions and tending to search those. Bayesian optimization is easy to use and according to works such as Shi et al. [20], can provide good performance and converge consistently on problems such as accelerator DSE and neural network-accelerator scheduling.

3.4 Gradient-based Optimization

When the objective function is differentiable, gradient ascent or descent can be directly applied to the search parameters to find desirable points. Compared to black-box optimization, gradient-based methods tend to run faster and scale to a larger number of inputs, as is demonstrated by the use of gradient-based optimization in training neural networks. Even if the objective function cannot be directly differentiated, it is possible to construct a differentiable surrogate objective function, for example using a neural network, that approximates the true objective function and apply gradient ascent or descent to the surrogate function. Hardware performance is one area where such techniques have been applied, as demonstrated by recent works that use differentiable surrogate models to optimize accelerator performance [2] and improve simulator accuracy [19].

3.5 Search Space Optimization

The hardware design to performance function is complex, since it depends not just on hardware parameters, but also on the workload being run and the way that workload maps to the given hardware design. Hardware design points also tend to be discrete, with some components only supporting multiples of a certain size or powers of two in some parameters. When this is the case, it may be beneficial to remap the search space to make it easier to search. VAESA [7] applies variational autoencoders to create a more amenable latent space for DSE, where both black-box and gradient-based optimization methods can perform more efficiently.

3.6 Reinforcement Learning (RL)

In the past few years, deep reinforcement learning (DRL) has advanced rapidly, evolving from gameplay benchmarks to solving massive-scale real-world optimization problems. It has been applied across domains, and naturally has been applied to hardware design space exploration as well. Recent works by Jiang et al. [10, 9] show how RL can be used to co-optimize the hardware design and neural network architecture. We are particularly interested in ConfuciuX [13], which combines RL and genetic algorithms to optimize hardware resource assignments. Specifically, we focus on the RL agent present in the system.

4 ConfuciuX

ConfuciuX [13] optimizes the number of processing elements (also known as multiply-accumulate units) and the size of the local buffer, which stores part of the inputs, weights, and output partial sums. The input to the optimizer is a set of layers making up a neural network, for example ResNet-50, and the dataflow of an existing industry or academic

deep learning accelerator, one of NVDLA [1], Eyeriss [3], or ShiDianNao [5]. ConfuciuX optimizes hardware parameters given one layer at a time, but utilizes information about what hardware parameters were found for previous layers in its observations.

4.1 Observations

In ConfuciuX, the observations are 10-dimensional and consist of: 7 dimensions defining shape of one layer, 2 dimensions describing the optimal PE and buffer sizes found at the previous layer, which are the previous actions, and 1 dimension representing the index of the current layer in the overall neural network (action status).

4.2 Actions

The two actions are the PE dimension, limited to a set of 12 possible values, and the buffer size, also limited to a set of 12 values.

4.3 Reward Function

Performance estimates on the target neural network layers are provided by MAESTRO [15], an analytical performance model for neural network accelerators. In ConfuciuX, in addition to latency and energy performance objectives, there is also a chip area constraint. Chip area is also estimated by MAESTRO. Since it is more expensive to produce larger chips, despite larger chips being generally more performant, it is important to put a constraint on the types of designs that can be generated. From [13], the reward function is as follows:

$$R = \begin{cases} P_t - P^{min}, \text{if} L^{budget} \geq 0 \\ Penalty, \text{otherwise} \end{cases} \quad (1)$$

According to the paper, P_t is performance on the current layer, P^{min} is the worst performance found so far (both are negative), L^{budget} is the area budget remaining, and $Penalty$ is the negation of all summed rewards in the current episode.

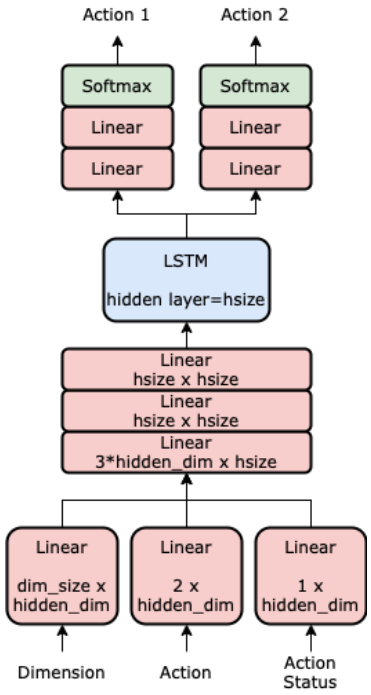


Figure 4: ConfuciuX policy network architecture

4.4 RL Algorithm

ConfuciuX uses REINFORCE as its underlying algorithm. Figure 4 depicts the original ConfuciuX policy network architecture. The policy network is a recurrent neural network (RNN) with long short-term memory (LSTM) architecture, where at each time step the next layer in the target network is examined. An LSTM is used in order to reuse information from past decisions made for other layers.

5 ConfuciuX-TF Architecture

The policy in ConfuciuX is a combination of fully-connected layers and an LSTM. We hypothesized that replacing the RNN-based policy network of ConfuciuX with a transformer-based model will find a hardware configuration with better rewards, i.e. the execution time to run a deep learning model. We have three inputs to the ConfuciuX policy network from the observations mentioned in section 4.1, which are the dimensions of the neural network model layers, the action values (the number of PEs and the size of buffers), and the action status (whether the action is done or not). As in figure

4, The original ConfuciuX policy network converts the inputs to three vectors of same size pass the concatenation of the three vectors through a number of linear layers to a size of hidden dimension, feed it to an LSTM, and then convert it to size of the action dimension with linear layers.

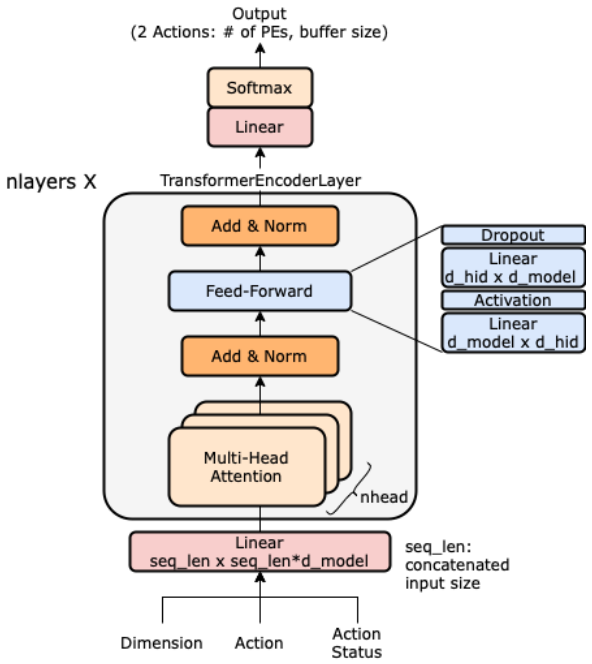


Figure 5: ConfuciuX-TF policy network architecture

We converted the original policy network to a combination of a transformer encoder and a smaller number of linear layers. The original policy network uses 10 linear layers and one LSTM, while our solution uses 2 linear layers and one transformer encoder. Usually, an embedding layer and a positional encoder is used to process the input before feeding it to a transformer network. We omitted the positional encoder because unlike tokens in a language model our inputs do not have a dependency or relation due to their relative positions. Also, we did not use an embedding layer to process our inputs since our three inputs with the same value should not be translated into the same embedding vector. Instead, we concatenated the three inputs and fed it to a linear layer.

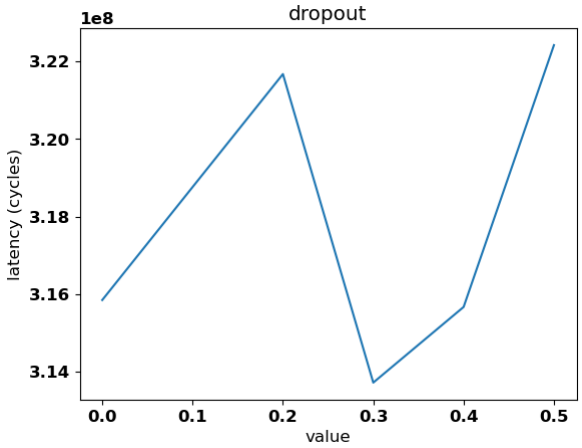


Figure 6: Hyperparameter search over dropout

6 Evaluation

We compared our ConfuciuX-TF solution and the vanilla ConfuciuX on a fixed random seed. Our evaluation result is described in table 2. The vanilla ConfuciuX baseline result differs from [13] because of the random seed.

6.1 Hyperparameter search

Before comparing our Confuciux-TF solution with the baseline, we performed a grid search of hyperparameters to find the optimal policy network that achieves the best optimization results (execution cycles).

First, we did a sweep over the dropout rate of the transformer encoder layers. Then, we did a grid search over the following four hyperparameters. We also did a simple sweep like what we did on the dropout rate over the following four hyperparameters before doing the grid search, but we found that searching over only one of these hyperparameters were not enough to find the best combination of the four to achieve the best latency.

1. d_model is equivalent to the dimension of the embedding vector if an embedding layer was used. d_model can be also seen as the input size of the transformer encoder.

2. d_hid is the dimension of the feedforward network in the transformer encoder layer.

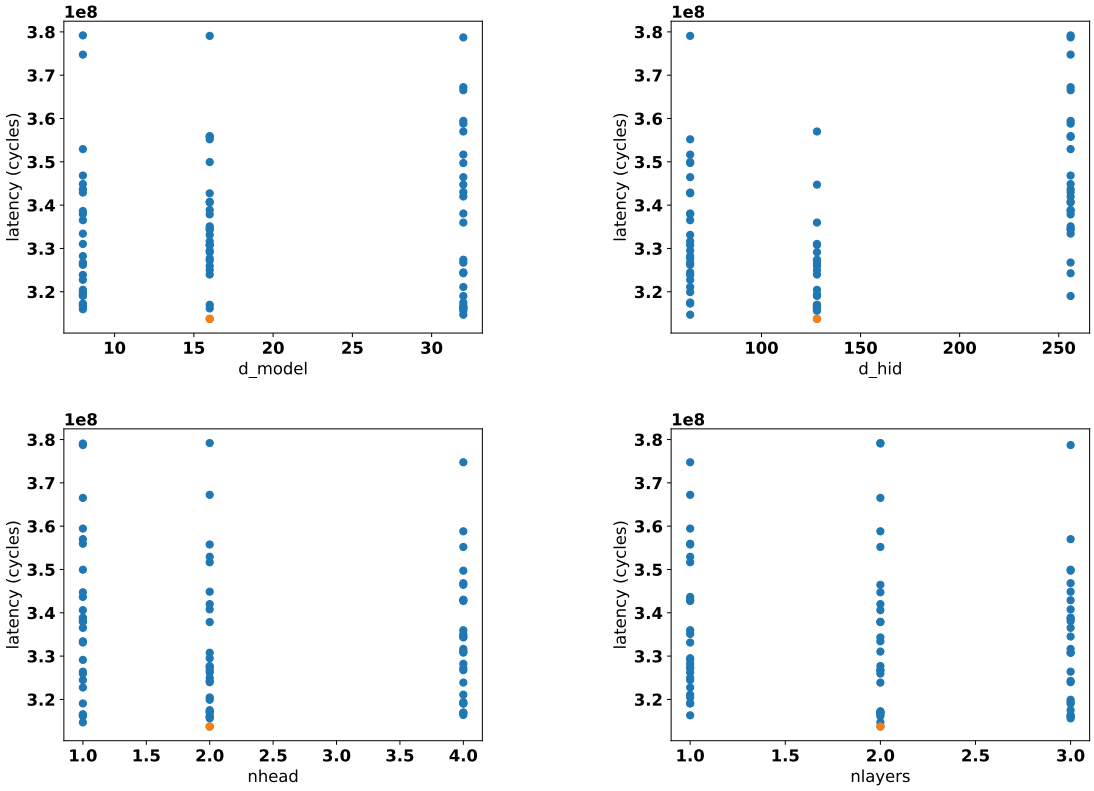
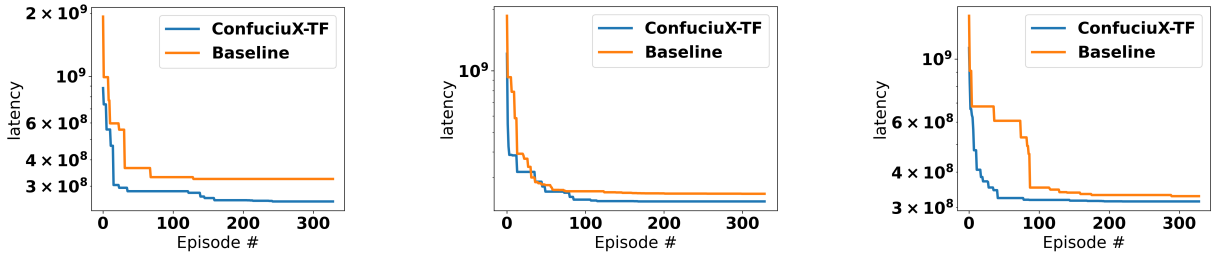


Figure 7: Scatter plots of all 81 hyperparameter search points evaluated for the optimal policy network, showing the distribution of points across one axis at a time. The orange point in each plot is the best searched set of hyperparameters, with settings d_model=16, d_hid=128, n_head=2, n_layers=2.



(a) Resnet-50, NVDLA (b) Resnet-50, Eyeriss (c) Resnet-50, ShiDianNao

Figure 8: Training curve

3. nhead is the number of heads in the multi-head attention model in the transformer encoder layer.

4. nlayers is the number of transformer encoder layers in the transformer encoder.

Our hyperparameter sweep results are depicted in figure 6 and 7. We tested each setting on Resnet-50 workload and ShiDianNao hardware architecture.

To begin with the dropout, we fixed the other four hyperparameters. Figure 6 shows that dropout of 0.3 is the best setting.

Each dot in figure 7 corresponds to the latency result of each setting, where smaller latency is better. The orange dot is our best setting, where d_model=16, d_hid=128, n_head=2, n_layers=2. We

Model	Architecture	Baseline	ConfX-TF	Improvement
ResNet-50	NVDLA	3.24E+08	2.53E+08	21.9%
ResNet-50	Eyeriss	2.49E+08	2.28E+08	8.3%
ResNet-50	ShiDianNao	3.29E+08	3.15E+08	4.2%
MnasNet	NVDLA	2.52E+08	2.44E+08	3.0%
MnasNet	Eyeriss	8.94E+08	6.39E+08	28.4%
MnasNet	ShiDianNao	10.21E+08	8.68E+08	15.0%
ShuffleNet-V2	NVDLA	3.24E+08	2.53E+08	21.9%
ShuffleNet-V2	Eyeriss	2.49E+08	2.28E+08	8.3%
ShuffleNet-V2	ShiDianNao	3.29E+08	3.15E+08	4.2%

Table 2: The result above is latency (execution time) for running the model on the hardware configuration found by the agent based on the (hardware) architecture. Baseline used vanilla ConfuciuX and our solution used ConfuciuX-TF. The latency is rounded to the nearest hundredth and the improvement in percentage is rounded to the nearest tenth.

will use these hyperparameter values to compare our solution to the baseline in the next section.

6.2 Agent Performance

We ran our solution and the baseline ConfuciuX with an epoch of 333 and we used RL-only option when running both agents. Our training curves in figure 8 prove that the epoch of 333 is enough to gather a converged solution per each run. As stated in [13], our search objective (reward) is latency and our constraint is area. We tested on different neural network workloads and different hardware architecture styles using the same experiment constraint as in table III of [13]. For example, Resnet uses a looser area constraint (Cloud in [13]) while MnasNet and ShuffleNet use tighter area constraints (IoT, IoTx in [13]). Our result is in Table 2.

Table 2 suggests that our solution outperformed the baseline in every model and hardware architecture style. The average latency improvement is 12.1%, which is a significant saving considering the scale of the execution time.

6.3 Runtime

Our ConfuciuX-TF solution not only achieves a better target metric, which is the latency to run a neural network model workload in this case, but also a better exploration time. Since ConfuciuX-TF replaced the LSTM cell of the policy in ConfuciuX with a transformer encoder, we can gain benefit from the running time. The time to run the baseline agent for 333 steps is in average 9.2 minutes while the time to run ConfuciuX-TF agent is about 6.6 minutes. More accurate running times on different model and architecture are described in table 3. The results in table 3 shows that our ConfuciuX-TF solution had a significant 27% runtime saving compared to the baseline ConfuciuX agent. This is a huge gain considering the scale and the iterative nature of hardware design space exploration.

Model, Arch	Baseline	ConfX-TF	Improv.
Resnet-50, NVDLA	12m 38s	9m 29s	24.9%
Resnet-50, Eyeriss	9m 20s	6m 10s	33.9%
Resnet-50, ShiDianNao	9m 34s	6m 41s	30.1%
MnasNet, NVDLA	11m 53s	8m 10s	31.3%
MnasNet, Eyeriss	8m 30s	5m 22s	36.9%
MnasNet, ShiDianNao	11m 9s	6m 33s	41.3%
ShuffleNet, NVDLA	6m 42s	5m 52s	12.4%
ShuffleNet, Eyeriss	6m 8s	5m 58s	2.7%
ShuffleNet, ShiDianNao	7m 2s	4m 57s	29.6%

Table 3: The time to run ConfuciuX-TF and ConfuciuX for 333 RL steps.

7 Conclusion

In this paper, we propose a transformer-based RL approach on hardware design space exploration. To be specific, we replace the RNN-based policy network of a prior RL-based work [13] with a neural network comprised mainly of transformer encoder

layers. We test our new agent in a similar setting to [13] to observe the overall improvement in terms of finding a more optimal hardware configuration for running deep learning workloads, in other words a hardware configuration that results in lower latency. We have shown that our proposed ConfuciuX-TF agent achieves 12.1 percent better latency on average compared to the baseline ConfuciuX agent.

Also, when we substitute the RNN with a transformer encoder of similar hidden layer size, the iteration time of the agent becomes shorter than that of the baseline ConfuciuX agent. Our experiments show that the runtime improvement is 27 percent on average. This means with our proposed solution, we can dramatically reduce the time to find a hardware configuration, which is a very iterative and tedious process. Considering the importance of automated design space exploration, this positive result introduces new opportunities for future usage.

References

[1] Nvidia deep learning accelerator (nvdla). `https://nvdla.org/`, 2017.

[2] ACHARARIT, P., HANIF, M. A., PUTRA, R. V. W., SHAFIQUE, M., AND HARA-AZUMI, Y. Apnas: Accuracy-and-performance-aware neural architecture search for neural hardware accelerators. *IEEE Access 8* (2020).

[3] CHEN, Y.-H., KRISHNA, T., EMER, J. S., AND SZE, V. Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks. *IEEE Journal of Solid-State Circuits 52*, 1 (2017), 127–138.

[4] DEEPAK NARAYANAN, MOHAMMAD SHOEYBI, J. C. P. L. M. P. V. K. D. V., AND CATANZARO, B. Scaling Language Model Training to a Trillion Parameters Using Megatron. `https://developer.nvidia.com/blog/scaling-language-model-training-to-a-trillion-parameters-using-megatron/`.

[5] DU, Z., FASTHUBER, R., CHEN, T., IENNE, P., LI, L., LUO, T., FENG, X., CHEN, Y., AND TEMAM, O. Shidiannao: Shifting vision processing closer to the sensor. In *2015 ACM/IEEE 42nd Annual International Symposium on Computer Architecture (ISCA)* (2015), pp. 92–104.

[6] HENNESSY, J., AND PATTERSON, D. John hennessy and david patterson deliver turing lecture at isca 2018, 2018.

[7] HUANG, Q., HONG, C., WAWRZYNEK, J., SUBEDAR, M., AND SHAO, Y. S. Learning a continuous and reconstructible latent space for hardware accelerator design. In *International Symposium on Performance Analysis of Systems and Software (ISPASS)* (2022).

[8] ÏPEK, E., MCKEE, S. A., CARUANA, R., DE SUPINSKI, B. R., AND SCHULZ, M. Efficiently exploring architectural design spaces via predictive modeling. In *Proceedings of the 12th International Conference on Architectural Support for Programming Languages and Operating Systems* (New York, NY, USA, 2006), ASPLOS XII, Association for Computing Machinery, p. 195–206.

[9] JIANG, W., YANG, L., DASGUPTA, S., HU, J., AND SHI, Y. Standing on the shoulders of giants: Hardware and neural architecture co-search with hot start. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* (2020).

[10] JIANG, W., YANG, L., SHA, E. H.-M., ZHUGE, Q., GU, S., DASGUPTA, S., SHI, Y., AND HU, J. Hardware/software co-exploration of neural architectures. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)* (2020).

[11] JONES, A., YAZDANBAKHSH, A., AKIN, B., ANGERMUELLER, C., LAUDON, J. P., SWERSKY, K., HASHEMI, M., NARAYANASWAMI, R., CHATTERJEE, S., AND ZHOU, Y. Apollo: Transferable architecture exploration. In *ML for Systems Workshop at NeurIPS 2020* (2020).

[12] JOUPPI, N. P., YOUNG, C., PATIL, N., PATTERSON, D., AGRAWAL, G., BAJWA, R., BATES, S., BHATIA,

S., BODEN, N., BORCHERS, A., BOYLE, R., CANTIN, P.-L., CHAO, C., CLARK, C., CORIELL, J., DALEY, M., DAU, M., DEAN, J., GELB, B., GHAEMMAGHAMI, T. V., GOTTIPATI, R., GULLAND, W., HAGMANN, R., HO, C. R., HOGBERG, D., HU, J., HUNDT, R., HURT, D., IBARZ, J., JAFFEY, A., JAWORSKI, A., KAPLAN, A., KHAITAN, H., KILLEBREW, D., KOCH, A., KUMAR, N., LACY, S., LAUDON, J., LAW, J., LE, D., LEARY, C., LIU, Z., LUCKE, K., LUNDIN, A., MACKEAN, G., MAGGIORE, A., MAHONY, M., MILLER, K., NAGARAJAN, R., NARAYANASWAMI, R., NI, R., NIX, K., NORRIE, T., OMERNICK, M., PENUKONDA, N., PHELPS, A., ROSS, J., ROSS, M., SALEK, A., SAMADIANI, E., SEVERN, C., SIZIKOV, G., SNELHAM, M., SOUTER, J., STEINBERG, D., SWING, A., TAN, M., THORSON, G., TIAN, B., TOMA, H., TUTTLE, E., VASUDEVAN, V., WALTER, R., WANG, W., WILCOX, E., AND YOON, D. H. In-datacenter performance analysis of a tensor processing unit. In *Proceedings of the 44th Annual International Symposium on Computer Architecture* (New York, NY, USA, 2017), ISCA '17, Association for Computing Machinery, p. 1–12.

[13] KAO, S., JEONG, G., AND KRISHNA, T. Confuciux: Autonomous hardware resource assignment for DNN accelerators using reinforcement learning. In *53rd Annual IEEE/ACM International Symposium on Microarchitecture, MICRO* (2020), IEEE, pp. 622–636.

[14] KARANDIKAR, S., MAO, H., KIM, D., BIANCOLIN, D., AMID, A., LEE, D., PEMBERTON, N., AMARO, E., SCHMIDT, C., CHOPRA, A., HUANG, Q., KOVACS, K., NIKOLIC, B., KATZ, R., BACHRACH, J., AND ASANOVIĆ, K. FireSim: FPGA-accelerated cycle-exact scale-out system simulation in the public cloud. In *Proceedings of the 45th Annual International Symposium on Computer Architecture* (Piscataway, NJ, USA, 2018), ISCA '18, IEEE Press, pp. 29–42.

[15] KWON, H., CHATARASI, P., PELLAUER, M., PARASHAR, A., SARKAR, V., AND KRISHNA, T. Understanding reuse, performance, and hardware cost of DNN dataflow: A data-centric approach. In *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture, MICRO* (2019), ACM, pp. 754–768.

[16] LI, Y., HAO, C., ZHANG, X., LIU, X., CHEN, Y., XIONG, J., HWU, W.-M., AND CHEN, D. Edd: Efficient differentiable dnn architecture and implementation co-search for embedded ai solutions. *arXiv preprint arXiv:2005.02563* (2020).

[17] LIN, Y., YANG, M., AND HAN, S. Naas: Neural accelerator architecture search. In *Design Automation Conference (DAC)* (2021).

[18] PARASHAR, A., RAINA, P., SHAO, Y. S., CHEN, Y.-H., YING, V. A., MUKKARA, A., VENKATESAN, R., KHAILANY, B., KECKLER, S. W., AND EMER, J. Timeloop: A systematic approach to dnn accelerator evaluation. In *2019 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)* (2019), pp. 304–315.

[19] RENDA, A., CHEN, Y., MENDIS, C., AND CARBIN, M. Difftune: Optimizing cpu simulator parameters with learned differentiable surrogates. In *Proceedings of the International Symposium on Microarchitecture (MICRO)* (2020).

[20] SHI, Z., SAKHUJA, C., HASHEMI, M., SWERSKY, K., AND LIN, C. Using bayesian optimization for hardware/software co-design of neural accelerators. In *Workshop on ML for Systems at the Conference on Neural Information Processing Systems (NeurIPS)* (2020).

[21] YANG, X., GAO, M., LIU, Q., SETTER, J., PU, J., NAYAK, A., BELL, S., CAO, K., HA, H., RAINA, P., KOZYRAKIS, C., AND HOROWITZ, M. Interstellar: Using halide's scheduling language to analyze dnn accelerators. In *Proceedings of the International Conference on Architectural Support for Programming Languages and Operation Systems (ASPLOS)* (2020).

[22] YAZDANBAKHSH, A., ANGERMUELLER, C., AKIN, B., ZHOU, Y., JONES, A., HASHEMI, M., SWERSKY, K., CHATTERJEE, S., NARAYANASWAMI, R., AND LAUDON, J. Apollo: Transferable architecture exploration. *arXiv preprint arXiv:2102.01723* (2021).