Physically Accurate Learning-based Performance Prediction of Hardware-accelerated ML Algorithms

Hadi Esmaeilzadeh
University of California San Diego
La Jolla, CA, USA

Soroush Ghodrati
University of California San Diego
La Jolla, CA, USA

Andrew B. Kahng
University of California San Diego
La Jolla, CA, USA

Joon Kyung Kim
University of California San Diego
La Jolla, CA, USA

Sean Kinzer
University of California San Diego
La Jolla, CA, USA

Sayak Kundu
University of California San Diego
La Jolla, CA, USA

Rohan Mahapatra
University of California San Diego
La Jolla, CA, USA

Susmita Dey Manasi
University of Minnesota
Minneapolis, MN, USA

Sachin S. Sapatnekar
University of Minnesota
Minneapolis, MN, USA

Zhiang Wang
University of California San Diego
La Jolla, CA, USA

Ziqing Zeng
University of Minnesota
Minneapolis, MN, USA

ABSTRACT

Parameterizable ML accelerators are the product of recent breakthroughs in machine learning (ML). To fully enable the design space exploration, we propose a physical-design-driven, learning-based prediction framework for hardware-accelerated deep neural network (DNN) and non-DNN ML algorithms. It employs a unified methodology, coupling backend power, performance and area (PPA) analysis with frontend performance simulation, thus achieving realistic estimation of both backend PPA and system metrics (runtime and energy). Experimental studies show that the approach provides excellent predictions for both ASIC (in a 12nm commercial process) and FPGA implementations on the VTA and VeriGOOD-ML platforms.

CCS CONCEPTS

• **Hardware → Physical design (EDA)**; **Hardware accelerators**; **Application specific integrated circuits**;

KEYWORDS

PPA prediction, design space exploration, ML accelerator

ACM Reference Format:
Hadi Esmaeilzadeh, Soroush Ghodrati, Andrew B. Kahng, Joon Kyung Kim, Sean Kinzer, Sayak Kundu, Rohan Mahapatra, Susmita Dey Manasi, Sachin S. Sapatnekar, Zhiang Wang, and Ziqing Zeng. 2022. Physically Accurate Learning-based Performance Prediction of Hardware-accelerated ML Algorithms. In *Proceedings of the 2022 ACM/IEEE Workshop on Machine Learning for CAD (MLCAD '22), September 12–13, 2022, Snowbird, UT, USA.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3551901.3556489

MLCAD '22, September 12–13, 2022, Snowbird, UT, USA.

ACM ISBN 978-1-4503-9486-4/22/09.
https://doi.org/10.1145/3551901.3556489

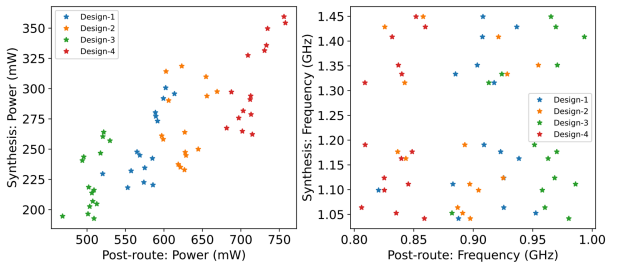
1 INTRODUCTION

Recent advances in machine learning (ML) algorithms have fueled a growing demand for application-specific ML hardware accelerators. The design of these accelerators is non-trivial: the design cycle from architecture to silicon implementation often takes months to years and involves a large team of cross-disciplinary experts. Under stringent time-to-market requirements, it is imperative to reduce turnaround time without sacrificing performance.

Recent research has developed automation flows for generating parameterizable FPGA- and/or ASIC-based accelerators. Parameterizable deep neural networks (DNNs) accelerators include VTA [3, 18], GeneSys [7], and Gemmini [8]. Accelerators for non-DNN ML algorithms [23], such as support vector machines or linear/logistic regression, have widespread applications, but have seen more limited research, with the TABLA platform [16] being a prominent example of a general-purpose non-DNN accelerator.

Generators such as those listed above allow a designer to configure key parameters of a DNN/non-DNN ML accelerator, e.g., the number of processing units or the on-chip memory configuration. The accelerator hardware description is then automatically translated to hardware at the register-transfer level (RTL). The search for an optimal configuration involves tradeoffs between the *power dissipation*, *performance*, and *area* (PPA) of the hardware platform, and the *energy* and *runtime* required to execute an ML algorithm on the platform. Therefore, this optimization involves the solution of two problems: (1) generating a hardware platform that optimizes the PPA metrics of the hardware, and (2) selecting a PPA-optimized hardware configuration that optimizes system-level metrics such as the runtime and energy required to run an ML algorithm.

The prediction of *platform PPA* based on an architectural description is a longstanding challenge in electronic design automation. In modern nanoscale technologies, the link between physical design and PPA is particularly acute; moreover, for many ML hardware platforms, a considerable fraction of the layout area is occupied by



Design	Kendall rank correlation coefficient for power	Kendall rank correlation coefficient for frequency
Design-1	0.6	-0.06
Design-2	0.09	-0.06
Design-3	0.35	0.09
Design-4	0.54	0.14

Figure 1: Miscorrelation of post-synthesis and post-routing metrics: total power and clock frequency for TABLA designs.

large memory macros which exacerbates the problem of PPA prediction. The prediction of *system-level metrics*, such as the runtime and energy required to execute an ML algorithm, is performed using system-level simulation engines. These simulators model data transfer and computation within the accelerator to determine the number of operations, stall cycles, memory latencies, etc. Since they use the frequency and power metrics of the hardware platform as inputs, their accuracy depends on the quality of PPA prediction.

Given an ML accelerator and a target clock period, the metrics of interest are the PPA of the hardware and the energy and runtime required to run ML algorithms. Optimizing PPA does not guarantee optimal runtime and energy consumption, e.g., smaller hardware consumes less power but requires more cycles to execute the ML algorithm, which may result in larger energy consumption. Design space exploration (DSE) finds an optimal architectural configuration and its hardware implementation, meeting designer-specified tradeoffs between PPA, runtime, and energy consumption. The large number of tunable architectural parameters in an ML accelerator results in a huge design space, and DSE requires a fast evaluator of PPA, runtime, and energy consumption for numerous architectural configurations in this space. A DNN accelerator may easily have 5-10 million instances, and conventional evaluators require several days of synthesis, place, and route (SP&R) runs to evaluate even a single configuration. Parallel evaluation runs offer scant relief due to limited compute resources and restrictions on EDA tool licenses. Using post-synthesis PPA without P&R is inadequate: Figure 1 shows poor correlation between the post-synthesis and post-SP&R results for TABLA designs, visually and through the Kendall rank correlation coefficient (τ) ($0 \Rightarrow$ no correlation, $\pm 1 \Rightarrow$ strong correlation). Similar miscorrelation is observed for other designs. The τ values of four VTA designs for total power are 0.61, −0.20, 0.07, 0.47 and for effective clock frequency are 0.45, −0.20, −0.16, 0.10.

Contributions: To enable full utilization of the power of parameterizable ML accelerator synthesis, we propose a physical-design-driven, learning-based prediction framework for hardware-accelerated ML algorithms. Our ML-based approach accurately estimates PPA and system performance. It overcomes the limitations associated with the high cost of exploring the design space by using a modest number of SP&R backend runs to train ML models to predict the performance of unseen ML accelerator designs. Our primary contributions are summarized as follows:

- Backend and System Predictors for ASICs with prediction errors below 10% (DNNs), 10% (small ML), and 6% (non-DNNs).
- Backend and System Predictors for FPGAs with average errors below 12% for a set of non-DNN ML algorithms.
- We use DNN-based ML accelerators (GeneSys [7] and VTA [3, 18]), and non-DNN-based accelerators (TABLA [16] and Axiline [7]), to validate our framework. All the designs go through full SP&R ASIC flow in a foundry 12nm enablement to generate the training and validation datasets. We demonstrate the application of our framework for accurate model-guided DSE.

2 RELATED WORK

Prior efforts have sought to predict power, performance, and area (PPA) at different stages of the design flow using two classes of predictors, based on analytical models and ML models. The works in [10, 12] introduce ML models and demonstrate significant improvement in PPA prediction over previous analytical models such as ORION [21] and McPAT [13] models. ML algorithms, and specifically neural networks (NNs), have made it possible to model high-level and gate-level designs to predict PPA more accurately for ASIC and FPGA implementations. Recent works include power metric prediction for high-level synthesis (HLS) [6] and PPA prediction for memory compilers [15]. These approaches indicate that ML models outperform analytical models, and are more convenient to train a wide range of ready-to-use ML models. To the best of our knowledge, no prior work builds ML models based on full backend SP&R for ASIC, as in this paper. In [2], an NN-based ML model is used to predict power and performance for different microarchitectures and to find Pareto-optimal design points for power and performance. An ML model to estimate power metrics in HLS, along with sampling-based techniques to prune the search space, is presented in [14]; these methods are used to find the Pareto frontier and Pareto-optimal designs for FPGA implementations.

There is limited prior work on early prediction of DNN performance. Aladdin [20] combines PPA-characterized building blocks with a dataflow graph representation to estimate performance, but does not incorporate the impact of physical design (PD) decisions beyond the block level. However, these decisions can substantially impact system performance. NeuPart [17] develops an analytical model to predict energy for computation and communication in a DNN accelerator. AutoDNNchip [22] proposes a predictor for energy, throughput, latency, and area overhead of DNN accelerators based on architectural parameters. It determines system-level performance metrics in an analytical-model-based coarse-grained mode and a runtime-simulation-based fine-grained mode, but has no clear engagement with backend design optimizations. On the other hand, it is well-understood that the performance of an ML accelerator is acutely dependent on the tradeoffs made in backend design. Numerous technology, methodology, and tool/flow effects must be comprehended, modeled and exploited – e.g., [1] shows that post-routing *HPWL* can be improved by about 15% with different settings of flow knobs. ML accelerators are considerably more complex than the test-cases used in [1], and can be expected to show even more overall

variation in post-routing outcomes due to tool/flow effects. In contrast to all previous works, our approach takes the effect of backend flows into account and effectively ties the prediction of *system-level* performance metrics to backend PPA.

3 DEFINITIONS

We formally define a set of key terms used in our work.

- **ML accelerator (design).** An ML accelerator or design is the RTL netlist created by a parameterizable ML hardware generator.
- **Workload.** A workload is a user-specified ML algorithm (or a set of algorithms) that runs on an ML accelerator. Due to the inherent structure of the computation, for a given network, the cost metrics for a workload – i.e., the energy and runtime of the accelerator – depend on the network topology and not on the specific input data.
- **Architectural parameters.** These are a set of parameters used by a parameterizable ML hardware generator to generate an ML accelerator. A **configuration** is a specific setting of architectural parameters. The parameterizable ML hardware generator can only generate one ML accelerator for a given configuration. This means there is a one-to-one mapping between ML accelerators and configurations for a parameterizable ML hardware generator.
- **Target clock period.** The target clock period is the clock period in the .sdc (Synopsys Design Constraints) file. The target clock frequency (f_{target}) is the reciprocal of target clock period.

For an ML accelerator:

- **Power (*P*)** is the sum of internal, switching, and leakage power, as reported by the SP&R tool after post-routing optimization.
- **Performance** is the effective clock frequency($f_{effective}$). This is the reciprocal of effective clock period, defined as the target clock period minus the worst slack reported by the SP&R tool after post-routing optimization.
- **Energy (*E*)** is the total energy required to run the user-specified workload on the ML accelerator. Given an ML accelerator and a workload, a simulator computes the energy based on the instruction mix and the post-SP&R performance/power metrics of the accelerator submodules.
- **Runtime (*T*)** is the time required to run the user-specified workload. For an ML accelerator and a specific workload, a performance simulator is used to compute the runtime.

4 DEMONSTRATION PLATFORMS AND SIMULATORS

We demonstrate our approach on the accelerator engines from four parameterizable open-source ML hardware generators,

TABLA [16] implements non-DNN ML algorithms such as linear regression, logistic regression, support vector machines (SVMs), back-propagation, and recommender systems.

GeneSys [7] executes DNNs using an $M \times N$ systolic array for GEMM operations such as convolution, and an $N \times 1$ SIMD array for vector operations such as ReLU, pooling, and softmax.

Axiline [7] builds hard-coded implementations of small ML algorithms (e.g., SVM, logistic/ linear regression) for training/inference.

VTA [3, 18] is a DNN accelerator where compute module includes a GEMM core for convolution, with a 2D array of PEs. VTA is integrated with Apache TVM [5], a deep learning compiler stack.

Integrating system simulations with backend data. The simulators are integrated with the backend analyses, where they receive PPA characteristics generated by our SP&R flow. The simulators used in our study are obtained from the GitHub repository of the VeriGOOD-ML project [24] and VTA hardware design stack [26]. For a specific hardware configuration point provided as an input, the PPA characteristics feed the simulator with data such as the clock frequency, energy per access for each of the on-chip buffers, and dynamic and leakage power of systolic and SIMD hardware components or GEMM and ALU hardware components. The performance statistics provided by the simulator are combined with these backend data from the SP&R flow to produce end-to-end runtime, energy, and power for execution of the ML algorithm.

5 OUR APPROACH

5.1 Problem Formulation

For a specific parameterizable ML hardware generator, we build machine learning models to solve the following problems:

Problem 1 (Backend PPA prediction): Train a model such that

$$(P, f) = BackendMLModel(\{x_1, x_2, \cdots, x_n; f_{target}\}) \quad (1)$$

where $\{x_1, \cdots, x_n\}$ is the user-specified configuration, f_{target} is the user-specified target clock frequency, P is the total power and f is the effective clock frequency.

Problem 2 (System metrics prediction): Train a model such that

$$(E, T) = SystemMLModel(\{x_1, x_2, \cdots, x_n; f_{target}\}) \quad (2)$$

where $\{x_1, \cdots, x_n\}$ is the user-specified configuration, f_{target} is the user-specified target clock frequency, E is the total energy consumption and T is the runtime.

Figure 2: Backend PPA and system-level metrics prediction flows.

Backend PPA prediction. We approach backend PPA prediction in a supervised manner. Figure 2 presents the flow for backend PPA prediction when $IsBackend$ is Yes. As shown in Figure 2, we first sample a subset of configurations from the architectural parameter space and use an ML hardware generator to generate the corresponding design. We then create a PPA curve by running the SP&R flow multiple times with different f_{target}. This is very time-consuming: for a typical GeneSys design with 1–2 million instances, it involves 63 SP&R runs, which takes between 7 and 14 license days of runtime. [1] These PPA curves are used as a dataset to train the $BackendMLModel$. Finally, the trained $BackendMLModel$ predicts backend metrics for unseen configurations and given f_{target}.

System metrics prediction. We adopt a similar approach for system metric prediction. Figure 2 presents this prediction flow when

[1] All of our SP&R runs use 8-thread runs on Intel Xeon server hardware with 2.4GHz clock frequency and 192GB or larger RAM. The P&R runtime for a single 1M-instance GeneSys design is 1 day, which is approximately 5.5 times the synthesis runtime.

IsBackend is *No*. Here the PPA curves generated from the SP&R flow are further fed into the simulator to create the energy and runtime curves, and these are used as a dataset to train the *SystemMLModel*. Finally, the trained *SystemMLModel* predicts system metrics for unseen configurations and given f_{target}. For clarity, the differences between the two prediction flows are highlighted with the red outline in Figure 2.

5.2 ML Models

As described in Section 2, several ML techniques have been applied to PPA prediction and/or DSE. Among all these techniques, we have found the following regression approaches to be most effective:

- **Gradient Boosted Decision Trees (GBDT)** utilize multiple decision trees as weak predictors. New trees are added sequentially to minimize the loss function during the training process.
- **Random Forest (RF)** also uses decision trees, but trains each tree independently using random samples of data. The final decision is generated based on the voting over a set of trees or from the average of the prediction generated by each tree.
- **Artificial Neural Network (ANN)** is a biologically-inspired model consisting of multiple neuron/node layers: an input layer, one or more hidden layers, and an output layer consisting of single node. The output of a node is a linear transformation of the previous layer input followed by a non-linear/linear activation function.
- **Stacked Ensemble** uses multiple "base learner" algorithms to outperform individual base learners. Training entails (i) training of multiple base learners (e.g., RF and GBDT models); and (ii) training of a second-level "metalearner" to find the optimal combination of the base learners as the stacked ensemble model.

We train GBDT, RF and ANN models using the open-source platform H2O [25]. H2O enables fast, distributed, in-memory, and scalable machine learning-based model training. We use the H2O AutoML grid search (based on random discrete [4] search algorithm) for hyperparameter tuning. This search method uniformly samples the hyperparameters from the given set and trains the model. The search process can be terminated based on the number of the trained models, runtime, or metric-based stopping criteria. We use maximum runtime-based termination in the hyperparameter tuning. Finally, we use these trained models as base learners and train the H2O stacked ensembles using linear regression. Theorem 1 in [11] proves that the stacked ensemble model will perform as the best learner asymptotically. Our framework utilizes the stacked ensembles to predict backend and system metrics. We describe the detailed steps to train the models in Section 6.

5.3 Two-Stage Inference Model

Design space exploration seeks an optimal implementation of an accelerator for a specified workload, to achieve better backend PPA and system metrics. To explore the design space more efficiently, we should pay more attention to a *region of interest* (ROI) in the design space, i.e., the region that contains the "optimal" implementation. Figure 3 gives an example of how to determine the ROI. In the example, we first use the Axiline platform to generate two accelerators (*Design-I* and *Design-II*) with different configurations, to implement a recommender system algorithm. We then run full SP&R flows for the two accelerators under 21 different f_{target} and compute corresponding energy and runtime system metrics. Figure 3(a) shows energy versus runtime of *Design-I* and *Design-II* for different f_{target}. We observe a typical division of the design space into three regions: (i) region of *runtime* where we can reduce the runtime at the cost of increasing energy; (ii) region of *balance* where we can achieve lowest energy without introducing too much runtime overhead; and (iii) region of *energy* where the energy will also increase when the runtime increases. In our work, we set the ROI to be the above region of balance. To visualize the ROI in terms of f_{target}, we show plots of runtime versus f_{target} in Figure 3(b). We can observe that the ROI excludes both extremely high and low f_{target}. Then, by examining $f_{effective}$ versus f_{target} (Figure 3(c)), we may further characterize the ROI in terms of the difference between f_{target} and $f_{effective}$: post-routed designs tend to have smaller negative slacks at higher f_{target} and larger positive slacks at lower f_{target}. Given the above, we define our ROI in terms of the difference between f_{target} and $f_{effective}$, as follows (ϵ is a parameter to determine the size of the ROI, with default value of 10%):

$$ROI = \{f_{target} | abs(f_{effective} - f_{target}) \leq \epsilon \times f_{target}\} \quad (3)$$

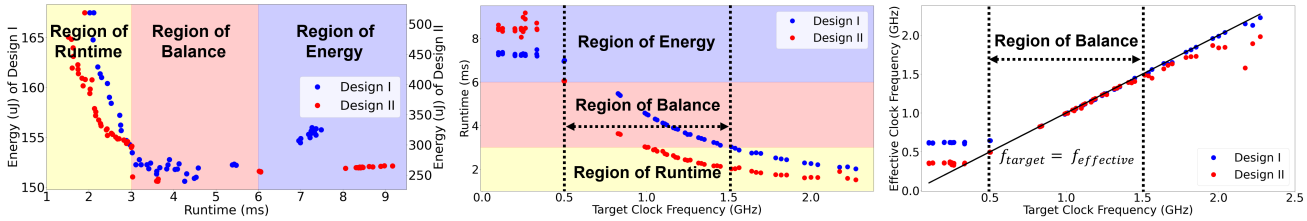
We further propose a *two-stage inference model* that is based on the above ROI definition. The model works as follows: (i) we first train models for backend PPA (power and performance) and system metrics (energy and runtime) independently; (ii) we invoke the trained performance model to predict the effective clock frequency $f_{effective}$ for a given target clock frequency f_{target}; (iii) if f_{target} and $f_{effective}$ satisfy Equation (3), we invoke trained power, energy and runtime models to generate corresponding metrics; (iv) otherwise, we abandon the given f_{target} and explore alternative f_{target}. The two-stage inference not only helps explore design space more efficiently, but also increases the prediction accuracy of our model. Our experimental results in Section 7.1 confirm the accuracy improvement of the two-stage inference model.

6 EXPERIMENTAL SETUP

6.1 Dataset Generation

Dataset generation includes three steps: (i) sampling architectural parameters and RTL generation; (ii) generating backend metrics using the ASIC or FPGA flow; (iii) generating system-level metrics.

(i) Sampling of architectural parameters and RTL generation. All platforms (TABLA, GeneSys, VTA and Axiline) are parameterizable, with corresponding tunable architectural parameters shown in Table 1. We use a variety of strategies to generate multiple configurations for each platform. Prior works on DNN accelerators based on systolic arrays [9, 19] and vector dot-products [3] report architectural parameters such as array dimension, data bitwidth, on-chip buffer size and off-chip bandwidth. For GeneSys and VTA, we use these insights from prior works to guide our choice of architectural parameters. We proportionally scale buffer size and bandwidth parameters based on array dimension. For each array dimension, we select other architectural parameters by changing the ratio of the buffer size to exercise various data reuse tradeoffs in DNNs. For example, a larger WBUF facilitates more weight reuse while a larger OBUF biases a design toward more on-chip reduction of the partial



(a) (b) (c)

Figure 3: Illustration for region of interest (ROI). (a) Energy versus runtime. The ROI is the pink region. (b) Runtime versus target clock frequency. Again, the ROI is the pink region. (c) Target clock frequency versus effective clock frequency. The ROI is the region where $f_{target} = f_{effective}$.

sums. For TABLA, we explore multiple configurations using variations of the structures shown in [16]. For Axiline, we use *Latin Hypercube Sampling* for integer architectural parameters such as dimension and number of cycles, thus achieving uniform coverage in each dimension; and we simply enumerate all the combinations for all other remaining architectural parameters.

Platforms	Feature	Candidate Values	Description
TABLA	PU	4, 8	# processing units
	PE	8, 16	# processing engines in each PU
	bitwidth	8, 16	bit width of internal bus
	input bitwidth	16, 32	bit width of *IO* bus
	benchmark	recommender systems backpropagation	ML algorithms
GeneSys	weight data width	4 – 8 (integer)	bit width of weight data (bit)
	activation data width	4 – 8 (integer)	bit width of input activation data (bit)
	accumulation width	32 (integer)	bit width of output accumulation (bit)
	WBUF capacity	16 – 256 (integer)	size of weight buffer (KB)
	IBUF capacity	16 – 128 (integer)	size of input buffer (KB)
	OBUF capacity	128 – 1024 (integer)	size of output buffer (KB)
	SIMD VMEM capacity	128 – 1024 (integer)	size of vector memory in VMEM (KB)
	WBUF AXI data width	64 – 256 (integer)	AXI bandwidth for the WBUF (bits/cycle)
	IBUF AXI data width	128 – 256 (integer)	AXI bandwidth for the IBUF (bits/cycle)
	OBUF AXI data width	128 – 256 (integer)	AXI bandwidth for the OBUF (bits/cycle)
	SIMD AXI data width	128 – 256 (integer)	AXI bandwidth for the VMEM (bits/cycle)
VTA	weight data width	8 (integer)	bit width of weight data (bit)
	activation data width	8 (integer)	bit width of input activation data (bit)
	accumulation width	32 (integer)	bit width of output accumulation (bit)
	WBUF capacity	16 – 256 (integer)	size of weight buffer (KB)
	IBUF capacity	16 – 128 (integer)	size of input buffer (KB)
	OBUF capacity	32 – 512 (integer)	size of output buffer (KB)
	off-chip bandwidth	64 – 512 (integer)	total external bandwidth (bits/cycle)
Axiline	benchmark	SVM, linear regression, logistic regression, recommender systems	ML algorithms
	bitwidth	8, 16	bitwidth for computation units
	input bitwidth	4, 8	bitwidth for initial inputs
	dimension	5 – 50 (integer)	dimension for stage 1 and 3
	num of cycles	1 – 32 (integer)	number of cycles required for stage 1 or 3 to process one input vector

Table 1: Architectural parameters for four design platforms.

(ii) Generation of backend metrics. We implement all the designs in a standard ASIC flow with GLOBALFOUNDRIES 12LP foundry enablement (Arm 9-track triple-VT cell libraries). We use Synopsys Design-Compiler-R-2020.09 to synthesize the netlist and Cadence Innovus 20.1 for place-and-route. To capture the entire PPA curve, we sweep the target clock period from 0.5ns to 10ns. In light of tool noise, we execute three SP&R runs for each target clock period by varying the target clock period by ±0.01ns. Backend PPA metrics (effective clock frequency $f_{effective}$ and power P) are extracted from DRC-clean post-routed layouts. We run 480, 344, 473 and 4558 SP&R runs for TABLA, GeneSys, VTA and Axiline designs respectively. We further implement TABLA designs on two different (Xilinx) FPGAs: Virtex-7 (xc7vx1140tflg1926-2G) and Kintex (xcku115-flva1517-1LV-i). Given the lower performance of these FPGAs compared to 12nm ASIC, we sweep the target clock period from 2ns to 30ns for each design. Similar to the ASIC flow, we run three runs for each target clock period to mitigate tool noise. A total of 630 data points are generated for each FPGA.

(iii) Generation of system metrics. We feed backend PPA metrics to simulators to obtain corresponding system metrics (energy E and runtime T). Here we set the widely used ResNet-50 and MobileNet-v1 networks as the workloads for GeneSys and VTA designs respectively. The workload of each TABLA or Axiline design is determined by its benchmark parameter (see Table 1).

6.2 Dataset Separation

An accelerator implementation depends on its (i) configuration, used by a platform to generate the accelerator; and (ii) target clock frequency, used by the SP&R flow to implement the accelerator in silicon. Thus, we evaluate our framework from two different aspects:

Predicting at unseen target clock frequency. Our framework should be able to predict backend and system metrics for known configurations at unseen target clock frequency. Hence, we split the dataset into training and test data by randomly shuffling and then uniformly sampling target clock frequencies. The training:test ratio is 4:1.

Predicting for unseen configuration. Our framework should predict backend and system metrics for unseen configurations. For this evaluation, we split the dataset into training dataset and test dataset based on configurations. The training:test ratio here is also 4:1.

6.3 Model Training

We train and evaluate a total of 192 ML models in this work.[2] For ASIC, there are four platforms (TABLA, GeneSys, Axiline and VTA), four metrics (power, performance, energy, runtime) and four ML model types (GBDT, RF, ANN and Stacked ensembles). For FPGA, there is one platform (TABLA),[3] two devices (Virtex, Kintex), and again, four metrics and four ML model types. The H2O package is used to autotune the hyperparameters of each model. The hyperparameters of each ML model type that we tune are shown in Table 2.

We first train the ANN, GBDT and RF models using H2O. We set the walltime of hyperparameter autotuning for GBDT, RF and ANN to 300s, 300s and 600s respectively. We use 16 CPU threads in parallel to accelerate the tuning process. Five-fold cross-validation is enabled and root mean squared error (RMSE) is used as the score

[2]For ASIC, there are 4 platforms × 4 metrics × 4 models × 2 separations of dataset, yielding 128 models. For FPGA, there are 2 devices × 4 metrics × 4 models × 2 separations of dataset, yielding 64 models.

[3]GeneSys designs, with millions of instances, are too large for our FPGAs.

function. The trained models with the best scores are used as the estimators to perform prediction on the test datasets. Then we use the grid search results of GBDT, RF and ANN models as base learners to train the stacked ensemble model. Here only the top seven models are chosen as the base learners to introduce enough variation while filtering out the poorly-performing models.[4]

Model	Parameters	Type	Range	Description
GBDT	n_estimator	integer	[20,500]	# gradient boosted tree
	max_depth	integer	[2,20]	maximum tree depth
RF	n_estimator	integer	[50 - 1000]	# decision trees in the forest
	mtries	*enum*	[1-total feature count]	# features considered for best split
	max_depth	integer	[5 - 100]	max tree depth
ANN	num_layer	integer	[1-4]	# hidden layers
	num_node	integer	[30-200]	# nodes in each hidden layer
	act_func	*enum*	[Tanh, Rectifier, Maxout]	activation function

Table 2: Tuned hyperparameters for *GBDT*, *RF* and *ANN*.

7 RESULTS AND DISCUSSION

7.1 ML Model Assessment

We use mean absolute percentage error (μAPE) and maximum absolute percentage error ($MAPE$) to evaluate our model.

Predicting at unseen target clock frequency. The results are presented in Table 3 and 4. Table 3 shows the performance of ML models for ASIC implementation (including TABLA, GeneSys, VTA and Axiline platforms) at unseen f_{target}. Table 4 shows the performance of ML models for FPGA implementations (TABLA platform only) at unseen f_{target}. We make the following observations regarding the performance of our ML models at unseen f_{target}:

- The reported μAPE for all four models are less than 5%, and the maximum $MAPE$ for the best model is about 20%.
- In order to compare the performance of different types of ML models (GBDT, RF, ANN and Ensemble), we first rank the four types of models from one to four based on the reported μAPE for each prediction task. Then we compute the average rank for each type of ML models by summing up its rank for each task, and dividing the sum of rank by the number of tasks. For the ASIC implementation, the average ranks for GBDT, RF, ANN and Ensemble models are respectively 2.6, 1.8, 3.8 and 1.7; for the FPGA implementation, the average ranks for GBDT, RF, ANN and Ensemble models are respectively 2.6, 1.6, 4.0 and 1.8. We can see that the RF and Ensemble models dominate for both ASIC and FPGA implementations.

Predicting for unseen configuration. Tables 5 and 6 show the performance of ML models for unseen ASIC (TABLA, GeneSys, VTA and Axiline) and FPGA (TABLA) implementations. Here we use black and red color to respectively indicate results of single-stage and two-stage inference models. We observe that:

- For ASIC implementations of unseen configurations across different platforms, the μAPE of backend PPA prediction and system metrics prediction is less than 10%.
- Our framework accurately predicts the backend and system metrics for FPGA implementation of unseen configurations. The μAPE of backend PPA prediction is less than 7% (5% in the corresponding ASIC flow) while the μAPE for system metrics prediction is less than 12% (6% in the corresponding ASIC flow). For the corresponding ASIC implementation of unseen TABLA configurations, the prediction error is higher due to the noisy outcome of the FPGA flow at higher f_{target}.
- Two-stage inference models achieve better (resp. worse) μAPE or $MAPE$ than single-stage models in 44 (resp. 28) "head-to-head" comparisons. This suggests our two-stage model can effectively filter out "uninteresting" noisy points in the design space.
- Similar to the prediction for unseen target clock frequencies, we also evaluate different types of ML models in terms of average rank, to compare their ability to predict for unseen configurations. For the ASIC implementation, the average ranks for GBDT, RF, ANN and Ensemble models are respectively 2.94, 2.38, 2.50 and 2.19; for the FPGA implementation, the average ranks for GBDT, RF, ANN and Ensemble models are respectively 2.13, 2.25, 3.50 and 2.13. We can see that Ensemble models dominate for both ASIC and FPGA implementations.

As no previous work builds a post-SP&R PPA prediction model for ML accelerators, no comparisons can be shown against prior work.

Table 3: Performance of ML models for ASIC implementation at unseen target clock frequencies.

Design	ML Model	Backend-Perf		Backend-Power		Sys-Runtime		Sys-Energy	
		μAPE	$MAPE$	μAPE	$MAPE$	μAPE	$MAPE$	μAPE	$MAPE$
TABLA	GBDT	3.55	36.09	2.59	11.82	3.68	17.35	2.44	14.36
	RF	3.25	17.43	2.67	10.59	**3.45**	11.04	**1.33**	8.09
	ANN	4.04	20.92	3.37	9.96	3.69	15.05	1.50	7.02
	Ensemble	**3.18**	14.43	**2.38**	10.93	3.63	15.03	1.45	7.60
GeneSys	GBDT	2.23	9.93	2.62	9.68	2.88	18.37	**0.74**	10.64
	RF	2.18	9.98	**2.30**	7.95	2.19	10.45	0.84	11.45
	ANN	3.23	13.83	5.28	18.81	3.80	13.71	4.17	27.65
	Ensemble	**1.94**	9.42	2.46	7.42	**2.18**	10.82	**0.74**	10.63
VTA	GBDT	2.03	9.15	2.26	8.91	2.23	12.05	**1.07**	7.20
	RF	1.96	11.60	2.19	15.17	**2.19**	11.38	1.37	8.70
	ANN	3.19	14.82	2.14	7.48	2.65	11.75	7.23	24.87
	Ensemble	**1.93**	9.63	**1.63**	8.65	2.24	10.36	1.23	7.23
Axiline	GBT	1.11	14.40	3.23	21.99	1.43	13.78	**1.05**	19.87
	RF	**1.05**	15.82	**1.50**	23.00	**1.24**	14.67	1.46	25.50
	ANN	1.79	16.93	4.47	34.60	1.51	15.24	3.83	21.79
	Ensemble	1.11	14.68	2.82	26.71	1.38	16.30	2.7	21.62

Table 4: Performance of ML models for different FPGA implementation (only TABLA) at unseen target clock frequencies.

FPGA	ML Model	Backend-Perf		Backend-Power		Sys-Runtime		Sys-Energy	
		μAPE	$MAPE$	μAPE	$MAPE$	μAPE	$MAPE$	μAPE	$MAPE$
Virtex-7	GBDT	**2.61**	11.91	2.54	13.43	2.91	9.27	1.64	7.55
	RF	2.74	13.00	**2.08**	11.26	**2.69**	12.72	**1.31**	8.96
	ANN	2.95	13.09	2.89	12.86	4.73	13.40	1.83	9.77
	Ensemble	2.71	12.86	2.39	14.73	2.83	11.95	1.35	7.72
Kintex	GBDT	2.77	15.80	1.38	7.16	2.54	11.86	1.77	9.34
	RF	2.76	17.80	**1.22**	7.69	2.70	13.21	**1.55**	7.19
	ANN	3.16	13.78	1.56	7.07	3.11	13.20	1.91	7.39
	Ensemble	**2.47**	12.24	1.32	6.21	**2.43**	9.59	1.74	8.95

7.2 Application of Our Approach

We apply our approach for design space exploration to two different platforms: Axiline and VTA.

7.2.1 Design space exploration for Axiline platform. We first use our approach to optimize the implementation of an accelerator that runs the support vector machine (SVM) algorithm with 200 features. The flow is as follows: (i) we choose two candidate configurations for architectural parameters of the Axiline platform (see Table 1); (ii) we use the Axiline platform to generate corresponding RTL netlists (denoted as **SVM-200 design1** and **SVM200 design2**);

[4] Experimental results suggest that choosing the top seven models gives the best results.

Table 5: Performance (better of single- and two-stage ML models) for ASIC implementation of different unseen configurations.

Design	ML Model	Backend-Perf		Backend-Power		Sys-Runtime		Sys-Energy	
		μAPE	*MAPE*	*μAPE*	*MAPE*	*μAPE*	*MAPE*	*μAPE*	*MAPE*
TABLA	GBDT	5.19	19.95	12.20	31.83	2.01	6.90	15.76	24.57
	RF	5.57	23.37	13.95	32.28	**1.27**	5.41	14.43	20.55
	ANN	5.70	27.29	**4.00**	12.08	4.28	9.70	**5.12**	13.85
	Ensemble	**4.76**	19.41	9.36	21.65	2.47	9.34	9.43	20.11
GeneSys	GBDT	**3.63**	12.27	5.70	14.88	7.33	14.16	16.77	23.81
	RF	3.72	13.55	4.48	11.82	**5.26**	11.77	16.23	19.87
	ANN	13.62	35.57	4.00	14.36	13.12	38.43	**9.62**	15.84
	Ensemble	4.62	14.19	**4.12**	8.07	6.28	20.27	15.93	18.25
VTA	GBDT	12.40	38.43	7.81	31.61	9.32	21.06	**2.75**	6.17
	RF	12.91	29.36	**4.99**	11.75	**5.85**	15.97	3.53	4.15
	ANN	10.82	27.36	8.28	26.25	6.98	20.44	15.35	31.29
	Ensemble	**9.71**	30.27	6.84	30.71	8.52	22.04	3.13	6.22
Axiline	GBDT	**1.60**	12.52	17.47	65.93	3.20	15.35	31.63	77.60
	RF	1.66	14.17	13.40	37.51	5.32	17.23	25.98	68.44
	ANN	1.95	11.92	**9.84**	40.98	**0.91**	5.16	**6.92**	21.51
	Ensemble	1.62	11.98	16.47	53.61	4.02	17.39	26.39	71.34

Table 6: Performance (better of single- and two-stage ML models) for FPGA implementation of different unseen TABLA configurations.

FPGA	ML Model	Backend-Perf		Backend-Power		Sys-Runtime		Sys-Energy	
		μAPE	*MAPE*	*μAPE*	*MAPE*	*μAPE*	*MAPE*	*μAPE*	*MAPE*
Virtex-7	GBDT	**2.73**	11.32	8.04	20.91	3.55	10.50	**10.34**	14.17
	RF	2.83	10.84	7.90	17.36	**3.40**	10.02	11.62	14.00
	ANN	4.76	24.15	7.65	18.37	4.04	9.34	13.34	23.07
	Ensemble	3.48	15.52	**6.57**	20.91	6.29	12.12	11.09	16.03
Kintex	GBDT	**3.13**	13.04	6.82	14.65	3.22	10.11	8.96	20.91
	RF	4.47	21.48	5.79	14.56	**3.07**	13.13	10.15	21.09
	ANN	5.61	33.30	3.79	8.67	4.63	11.05	12.36	26.80
	Ensemble	3.75	22.40	**4.86**	9.91	3.29	13.63	**8.55**	16.65

(iii) we invoke the trained machine learning model to predict system metrics (runtime and energy) for both designs under 32 different target clock frequencies. The results are presented in Figure 4.

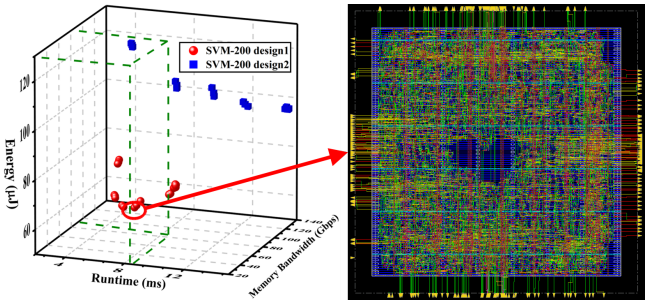


Figure 4: An application example of our framework. Left: Design space exploration for Axiline platform. Right: Layout for the SVM-200 design with the lowest predicted energy.

The memory bandwidth is calculated as: $memory\_bandwidth = \frac{num\_input\_vector \times input\_bitwidth}{runtime}$, where *input_bitwidth* is an architectural parameter of the Axiline platform and *num_input_vector* is the number of input vectors of the workload. Based on this exploration, we first circumscribe the set of points that meet system-level specifications on runtime and memory bandwidth by the green box shown in the 3D plot at left in Figure 4. We then select the lowest-energy point within the box using the energy prediction from trained ML models. Finally, we run a full SP&R flow at this point to implement the "optimal" accelerator. The full layout for this "optimal" accelerator is shown in the right half of Figure 4. Exhaustive ground-truth analysis with full SP&R for all 72 points confirms that all predictions for runtime and energy, evaluated in less than 1s are within 5% of post-SP&R values. The best solution from our ML models matches exhaustive implementation, which takes over 24 hours.

7.2.2 Design space exploration for VTA platform. We use our framework to perform DSE to search for an optimal implementation of an accelerator that implements MobileNet-v1 architecture using a similar framework as was used for the Axiline platform above. The results are presented in Figure 5. We first circumscribe the set of points that meet system-level specifications for runtime and energy by the green box shown in the 3D plot at left in Figure 5. We then find that *VTA design2* with target clock frequency of 50 MHz has the least predicted runtime and the lowest predicted energy. The full layout for this "optimal" accelerator is shown on the right half of Figure 5. Exhaustive ground-truth analysis with full SP&R for all 126 points confirms that the best solution identified by our ML models matches what exhaustive implementation would have found.

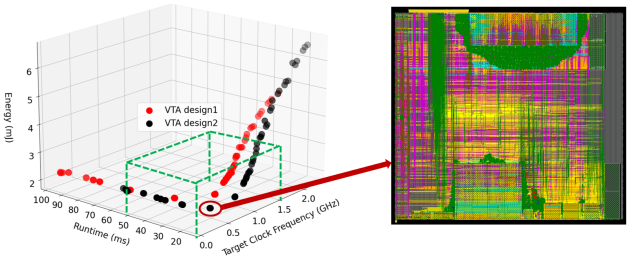


Figure 5: An application example of our approach. Left: Design space exploration for VTA platform. Right: Layout for the accelerator with the least predicted runtime and the lowest predicted energy.

8 CONCLUSION

We describe a physical-design-driven ML-based framework that achieves high prediction accuracy for both backend PPA metrics and system-level (energy and runtime) performance metrics. Our framework incorporates ML modeling aspects such as focus on a "region of interest", a novel two-stage inference approach, and the use of stacked ensemble models to enable efficient model-guided search over large configuration spaces of accelerator architectures for a given workload or ML algorithm. Our framework is extensively validated on multiple ML architecture platforms, and both ASIC and FPGA contexts.

9 ACKNOWLEDGMENTS

This material is based on research sponsored in part by Air Force Research Laboratory (AFRL) and Defense Advanced Research Projects Agency (DARPA) under agreement number FA8650-20-2-7009. Andrew B. Kahng is also supported in part by NSF CCF-2112665. The U. S. government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright notation thereon. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of AFRL, DARPA, or the U. S. government.

The authors would like to acknowledge the contributions of Steven M. Burns and Anton A. Sorokin from Intel Labs.

REFERENCES

[1] A. Agnesina et al., "VLSI Placement Parameter Optimization using Deep Reinforcement Learning", *Proc. ICCAD*, 2020, pp. 1-9.

[2] C. Bai et al., "BOOM-Explorer: RISC-V BOOM Microarchitecture Design Space Exploration Framework", *Proc. ICCAD*, 2021.

[3] S. Banerjee et al., "A Highly Configurable Hardware/Software Stack for DNN Inference Acceleration", *arXiv:2111.15024*, 2020.

[4] J. Bergstra and Y. Bengio, "Random Search for Hyper-parameter Optimization", *Journal of Machine Learning Research*, 13(10), 2012, pp. 281-305.

[5] T. Chen et al., "TVM: An Automated End-to-End Optimizing Compiler for Deep Learning", *Proc. OSDI*, 2018, pp. 578–594.

[6] S. Dai et al., "Fast and Accurate Estimation of Quality of Results in High-Level Synthesis with Machine Learning", *Proc. FCCM*, 2018, pp. 129-132.

[7] H. Esmaeilzadeh et al., "VeriGOOD-ML: An Open-Source Flow for Automated ML Hardware Synthesis", *Proc. ICCAD*, 2021, pp. 1-8.

[8] H. Genc et al., "Gemmini: Enabling Systematic Deep-Learning Architecture Evaluation via Full-Stack Integration", *Proc. DAC*, 2021, pp. 769-774.

[9] N. P. Jouppi et al., "In-datacenter Performance Analysis of a Tensor Processing Unit", *Proc. ISCA*, 2017, pp. 1–12.

[10] A. B. Kahng et al., "ORION3.0: A Comprehensive NoC Router Estimation Tool", *IEEE Embedded Systems Letters* 7(2) (2015), pp. 41-45.

[11] M. J. van der Laan et al., "Super Learner", Statistical Applications in Genetics and Molecular Biology. 2007; 6(1). https://doi.org/10.2202/1544-6115.1309

[12] W. Lee et al., "PowerTrain: A Learning-based Calibration of McPAT Power Models", *Proc. ISLPED*, 2015, pp. 189-194.

[13] S. Li et al., "McPAT: An Integrated Power, Area, and Timing Modeling Framework for Multicore and Manycore Architectures", *Proc. MICRO*, 2009.

[14] Z. Lin et al., "HL-Pow: A Learning-Based Power Modeling Framework for High-Level Synthesis", *Proc. ASP-DAC*, 2020, pp. 574-580.

[15] F. Last and U. Schlichtmann, "Feeding Hungry Models Less: Deep Transfer Learning for Embedded Memory PPA Models : Special Session", *Proc. MLCAD*, 2021, pp. 1-6.

[16] D. Mahajan et al., "TABLA: A Unified Template-based Framework for Accelerating Statistical Machine Learning", *Proc. HPCA*, 2016, pp. 14-26.

[17] S. D. Manasi et al., "NeuPart: Using Analytical Models to Drive Energy-Efficient Partitioning of CNN Computations on Cloud-Connected Mobile Clients", *IEEE TVLSI* 28(8) (2018), pp. 1844-1857.

[18] T. Moreau et al., "A Hardware–Software Blueprint for Flexible Deep Learning Specialization", *IEEE Micro*, 39(5) (2019), pp. 8-16.

[19] S. D. Manasi, and S. S. Sapatnekar, "DeepOpt: Optimized Scheduling of CNN Workloads for ASIC-based Systolic Deep Learning Accelerators", in *Proc. ASPDAC*, 2021, pp. 235-241.

[20] Y. S. Shao et al., "Aladdin: A Pre-RTL, Power-Performance Accelerator Simulator Enabling Large Design Space Exploration of Customized Architectures", *Proc. ISCA*, 2014, pp. 97-108.

[21] H. Wang et al., "Orion: A Power-Performance Simulator for Interconnection Networks", *Proc. MICRO*, 2002, pp. 294-395.

[22] P. Xu et al., "AutoDNNchip: An Automated DNN Chip Predictor and Builder for Both FPGAs and ASICs", *Proc. FPGA*, 2020, pp. 40-50.

[23] E. Tabanelli et al., "DNN Is Not All You Need: Parallelizing Non-Neural ML Algorithms on Ultra-Low-Power IoT Processors", *arXiv:2107.09448*, 2021. https://arxiv.org/abs/2107.09448.

[24] VeriGood-ML, https://github.com/VeriGOOD-ML/public.

[25] AutoML: Automatic Machine Learning, https://docs.h2o.ai/h2o/latest-stable/h2o-docs/automl.html.

[26] "VTA Hardware Design Stack", https://github.com/pasqoc/incubator-tvm-vta.