Fast and Accurate PPA Modeling with Transfer Learning

Luis Francisco
Department of Electrical and Computer Engineering
North Carolina State University
Raleigh, NC 27606 USA
lsfranci@ncsu.edu

Paul Franzon
Department of Electrical and Computer Engineering
North Carolina State University
Raleigh, NC 27606 USA
paulf@ncsu.edu

W. Rhett Davis
Department of Electrical and Computer Engineering
North Carolina State University
Raleigh, NC 27606 USA
wdavis@ncsu.edu

2021 ACM/IEEE 3rd Workshop on Machine Learning for CAD (MLCAD) | 978-1-6654-3166-8/21/$31.00 ©2021 IEEE | DOI: 10.1109/MLCAD52597.2021.9531109

***Abstract*—The power, performance, and area (PPA) of a System-on-Chip (SoC) is known only after a months-long process. This process includes iterations over the architectural design, register transfer level implementation, RTL synthesis, and place and route. Knowing the PPA estimates for a system early in the design stages can help resolve tradeoffs that will affect the final design. This work presents a machine learning approach using gradient boost models and neural networks to fast and accurately predict the PPA. This work focuses on reducing the number of samples used to create the models. The models use transfer learning to predict the PPA for new design configurations and corner conditions based on previous models. The models predict the PPA as a function of parameters accessible during the RTL synthesis. The proposed models achieved PPA predictions up to 99% accurate and using as few as 10 data samples can achieve accuracies better than 96%.**

***Index Terms*—PPA, Machine Learning, Power, Performance, Area, Gradient Boost, Neural Network, Transfer Learning.**

I. Introduction

Power, performance, and area (PPA) is needed in all the design stages of a system-on-Chip (SoC), but it is known with certainty only after the design process is complete. As this process can take months or even years for an entire chip, missing a PPA target can create significant delays in the design process. To accelerate the PPA estimation, we are using machine learning data-driven models to expedite the process. We can learn from previous chip designs to help accelerate this process even more.

Having an estimate of the PPA in the early design stages can help make critical decisions that have the greatest potential to optimize power. There is also a need to accurately predict power well enough to aid power delivery and thermal power management design. A PPA estimation model with the flexibility to be used in a system-level framework is highly desired. Another desired feature of a PPA model is that it can depend on higher-level architecture parameters.

In this work, we create fast and accurate PPA models using gradient boost and neural network algorithms. In PPA modeling, the most time consuming step is gathering the data to create such models. Most of the previous work in this area focuses on the prediction of a single design or process corner. In this work, we optimize our solution to require a minimum amount of data by using transfer learning to help train a model for new designs and multiple corner operating conditions.

II. State-of-the-art of PPA with Machine Learning

Power, performance, and area have been explored using different techniques over time. We can find approaches to estimate PPA in all design stages, including high-level synthesis, RTL, place and route.

In terms of machine learning (ML), the work in [1] uses a convolutional neural network (CNN) approach for power estimation. The CNN is trained with power simulations of an RTL/SystemC model for a RISC-V core. Other research for PPA estimation that uses ML are in [2], [3]. [3] uses clustering and neural networks to classify different power/performance profiles. A similar approach is used in [2] but for heterogeneous systems, including multiple core frequencies and memories. [4] presents an ML model for power estimation using high-level simulations of an IP block.

The work in [5] focus on predicting the dynamic average power. This work uses a graph neural network to generate a vector-based power estimation in a given number of clock cycles window. A power, performance, and area prediction (PPA) predictor for memories using neural networks is presented in [6]. This work is extended to use transfer learning and move parameters from a memory compiler model to another [7].

An approach for design space exploration using high-level synthesis parameters for a specific design can be found in [8]. This work not only focuses on PPA but provides insights to change parameters in a C++ template to deliver an optimal design. Another work for design exploration in high-level synthesis is presented in [9]. The research in [9] uses a neural

network (NN) with transfer learning to explore the quality of a design based generated by a high level synthesis tool.

The research in [10], introduces a machine learning framework to predict dynamic power in field programmable gate arrays (FPGA). This framework uses high-level synthesis and feature extracted test benches to generate switching activity data. They can create models with an average maximum absolute error of 9%.

In contrast with the previous works described in this section, we focus on creating PPA models with a reduced number of samples. We provide different modeling alternatives that can use high-level parameters and synthesis parameters. We also focus our efforts to obtain PPA models with at least 95% accuracy. We want those models to have the ability to predict new designs and corner conditions based on models created for previous designs and corner conditions.

III. PPA Modeling with Machine Learning

In this work, we propose the use of machine learning to predict the PPA. Since gathering data is costly, we focus on modeling algorithms that require a small number of samples. We explore different ML algorithms like neural networks, decision trees, and surrogate models. As we will explain later, we obtain better results with gradient boost regression (GB) and neural networks (NN). For this reason, we focus on GB and NN. We also implement a shared weights strategy with a neural network to transfer what the model learns from a design to new designs.

A. Gradient Boost Regressor

Gradient boosting can work in both regression and classification problems. It creates a final prediction based on an ensemble of weak predictions. Usually, the weak predictions are in the form of decision trees [11]. In machine learning algorithms we need to optimize a loss function $L\left(y, F\right)$ and store the weights from a learning kernel $h\left(x, \lambda\right)$. The gradient boost algorithm suggests to create a new kernel function $\alpha h\left(x_i, \lambda\right)$ parallel to the negative gradient $g_m\left(x_i\right)$ [11], [12]. This gradient is given by,

$$g_m\left(x_i\right) = \left[\frac{\partial L\left(y_i, F(x_i)\right)}{\partial F(x_i)}\right]_{F(x)=F_{m-1}(x).} \tag{1}$$

As the new kernel function and the loss function are highly correlated, we can reduce the loss optimization problem to a least-squares minimization problem:

$$\left(\lambda_m, \alpha_m\right) = \arg\min_{\lambda,\alpha} \sum_{i=1}^{N} \left[-g_m\left(x_i\right) - \alpha h\left(x_i, \lambda\right)\right]^2. \tag{2}$$

Finally, we can approximate the next update in the regression target function to be

$$F(x_m) = F_{m-1}\left(x\right) + \lambda_m h\left(x, \alpha_m\right). \tag{2}$$

A very efficient scalable implementation of gradient boost is XGBoost [13]. XGBoost uses a more accurate approximation of the gradients. Instead of the second partial derivatives of the loss function, it uses second-order gradients. XGBoost is a sparsity-aware algorithm, for sparse data tends to perform well. In this work, we assess both a standard gradient boost implementation and XGBoost.

B. Neural Network with Transfer Learning

A neural network is a set of connected nodes or neurons. Each connection is associated with a weight that, when trained, learns from the data. The neural network learns by setting those weights to extract different representations of the input data. Each layer requires an activation function [14]. In the NN models we are proposing we use a rectifier linear unit (ReLU) function for the activation for the hidden layers and a linear function for the output layer.

Since the learning is stored in terms of the weights, a NN can share the learning in similar problems. This means that if we train a NN for a specific problem, it is possible to share some of the layers to a problem that has similar features. We propose a model structure that consists of a base model where a set of weights are shared plus a set of layers trained for each unique problem (see Figure 1). This allows us to create future PPA predictions of new designs and predict the operating corners among designs.

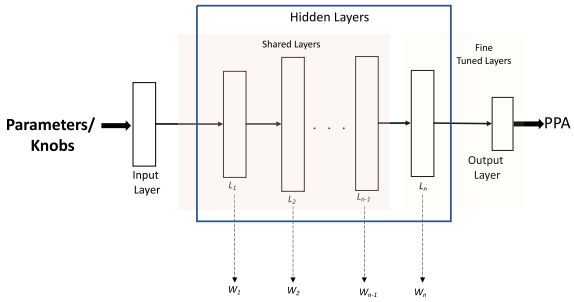

Fig. 1. Neural network with shared weights.

With the model in Figure 1, we can have a base model created for one design and re-train only a subset of weights for each design. This sharing will allow a reduction in the amount of data needed to develop new PPA predictions. In the same way, we can predict multiple operation corners for a design. We base this approach on the fact that the internal design structures are similar in a core or an IP block design. If we analyze the RTL, there will always be combinatorial and sequential logic blocks.

C. Other ML Algorithms Evaluated in this PPA Prediction

In addition to neural network and gradient boost models, we evaluate other decision trees and surrogate models. These algorithms are then compared with the other proposed models. As we will explain in the results, GB and NN are the best models we found.

In terms of **surrogate model** we consider four approaches implemented in [15]. These can be considered more traditional response surface models: radial basis function interpolant

with linear (RBFLinear) and cubic (RBFCubic), Gaussian process regressor (GP) and a third-order traditional polynomial regressor (Poly).

For other decision **trees or ensemble models**, we considered four additional models implemented in [16]: A bagging regressor (BR), AdaBoost regressor (AB), a decision tree (DTR) and a random forest regressor (RF).

IV. Model Parameters and Model Generation

A. Parameters Used for the PPA Models

Possible parameters include those related to architectural design, register-transfer level (RTL) implementation, RTL synthesis and place and route (PR). We will be focusing on modeling the power using RTL synthesis parameters and architectural parameters. In the case of memory we will also include memory organization parameters.

We selected synthesis parameters that can affect the gate-level netlist: clock period (Tclk), maximum transition time at a logic node (MaxTran), maximum fan-out at a logic node (Fanout) and clock uncertainty.

We have parameters that can affect the RTL code, those are considered architectural parameters. The architectural parameters are categorical parameters and the data generation with those parameters is very limited. In this case we are only using: numbers of function-units and L1 cache configurations.

When modeling the PPA for memories, we consider the next set of parameters that affects the memory generator. Those parameters will affect the memory organization. We can have similar memory size but arranged differently: number of addressable locations (WordDepth), data bits per word (8 to 256) (IOWidth), number of bit-cells that share a column mux and data-pin (Mux), cycles of latency due to pipelining and number of redundant columns for built-in self-repair.

We are modeling the PPA as a four components vector for logic designs: dynamic power, leakage power, minimum clock period period and area. The dynamic is represented as dynamic energy i.e., power multiplied by the clock period.

For the memories we have a five components vector that includes: leakage power, dynamic read power, dynamic write power, access time and area.

B. Model Creation and Data Collection Framework

We developed an interactive framework that can be used to generate the model in an expandable and automated way. This framework supports the model creation with synthesis parameters or architectural parameters; it also should be expandable to any IP or memory generator. Figure 2 illustrates the framework operation flow.

The model generation framework can take as input a design RTL, a memory generator or a core generator. Using a range of input parameters creates a sampling space using Latin hypercube (LHC) and runs the core generator, memory generator, or synthesis tool to extract the PPA. We can train the model on the fly with those samples or store the dataset.

Choosing the right samples can significantly affect the model accuracy and the design space that we can explore. We create the parameters samples space using the Latin hypercube (LHC) sampling strategy [17].

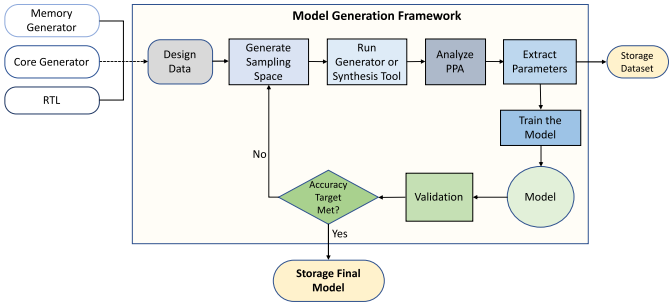

Fig. 2. Model and data generation framework.

V. Dataset to Create and Evaluate Models

We created a dataset consisting of parameterized cores and IP blocks (logic datasets). This dataset is used to train and validate the proposed modeling approaches. The dataset also includes memory macros from a memory generator (SRAM data). We do not have the L1 caches in the cores; that is the main reason to include separate SRAM data.

The logic datasets consist of RTL code and Synthesis-only results for power, performance, and area for several Rocket Core Configurations (L1 Cache not included) and one other core (OpenRISC 1200). This dataset also includes a convolutional neural network (CNN) accelerator IP block. The Rocket is an open-source parameterized core generator; it generates synthesizable RTL. We used a commercial $22nm$ technology with a reference methodology for synthesis.

We can see significant variations in the different Rocket configurations. Figure 3 shows the area as a function of the performance for the five configurations. The plot illustrates the immensity of the challenge for a designer to figure out the settings. We can also see variations for the leakage power and the area in Figure 4. As expected, leakage power is highly correlated with the area.

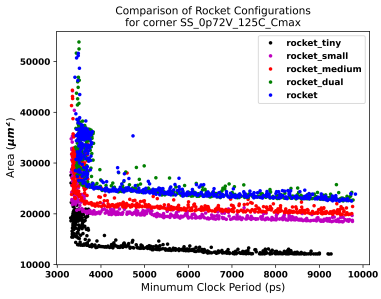

Fig. 3. Area vs. performance for the Rocket core variations.

The dataset also contains power, performance, and area under five unique corner conditions. Figure 5 shows how the area and the performance vary for all the corners conditions. We can see the variations in the corners conditions, especially for the low clock periods.

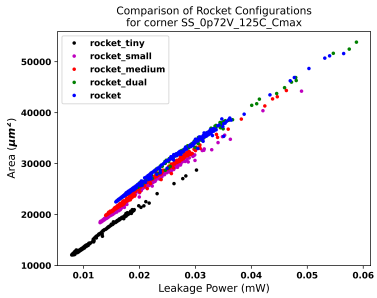

Fig. 4. Area vs. Leakage for the Rocket core variations.

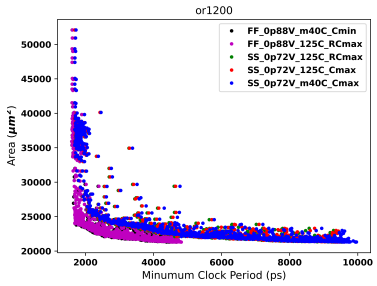

Fig. 5. Area vs. performance for OR1200 for 5 corner conditions.

VI. Experimental Results

A. Evaluating ML Model Formulations for PPA

We created the PPA models with the logic cores and IP block and the SRAM using the datasets created. Initially, those models were created using 500 samples, of which 80% was used to train and validate the models. The model was tested with the remaining 20% for a total of 100 test points.

We evaluate each model base on the overall accuracy given in terms of the mean absolute error (MAE):

$$MAE = \frac{100}{N}\sum_{i=1}^{N} abs\left(\frac{y_i - \hat{y_i}}{y_i}\right). \quad (2)$$

We also get the standard deviation of the absolute error. This allows us to evaluate the time required to train the models and to test new samples. We measure those times relative to the shortest time. We choose to do this instead of iterations because each model approach has a different target function. We analyze the accuracy by PPA component and the overall accuracy.

Table I shows the models results for the Rocket core default configuration. This table shows that the XGBoost model achieves the highest overall accuracy with the lowest standard deviation. We verified with multiple designs that in general, the boosting algorithm performs especially well in terms of accuracy. If we analyze the training and testing time, the XGB compares well to the smallest training and test time for the decision tree regressor (DTR). The absolute training times range from a few milliseconds to five minutes, and the testing time per 100 points is in the fractions of microseconds. The neural network, as expected, has the highest training, highest testing cost, and lower accuracy, but still over 97%. We did a similar analysis for the rest of the designs in the dataset with similar results.

When analyzing the training and testing time, we need to emphasize that we are competing with tools that take hours to generate a single sample for the testing time. Also, the training time is required only once per design. In Table I the training time goes from 5 minutes to as low as less than 5 milliseconds.

By using the five parameters previously described, we created the PPA models for the one port SRAM. This memory goes from $2K$ to $99K$ bits. Same as for the logic cores, we used the total samples in the dataset $(8,248)$ split in 80% for validation and for the remainder of testing.

Table II shows the model evaluation results for the memories. We can see that the gradient boost provides good accuracy from those results, but the best accuracy comes from the neural network and a low standard deviation. If we compare the model error of our models to some state-of-the-art models with 3%, our best model has an error lower than 0.5%. The models here have a better fit, but we also used more samples for the training. In training, the neural network will take longer to train but has better evaluation time than some of the tree algorithms.

B. Comparing PPA Models With the Number of Samples

After creating models to predict the power, performance and area accurately, we focused on reducing the number of samples needed to generate the models. To achieve this goal, we analyzed how the accuracy of the models is affected when we reduce the number of training data points. Having a model with a reduced number of samples will make it useful in practical chip design applications.

Figure 6 shows how the accuracy of the models changes for different training sample counts in the Rocket core default configuration. From this figure, we can see that the gradient boost is the one with the best performance. At the 10 samples mark, the surrogate models achieve an accuracy of 88.5% to 90.7. At this mark, the gradient boost algorithms achieve and accuracy of up to 95.1%. XGB gets this best accuracy. As expected with such a low number of samples the NN gets the lowest accuracy of 79.7%. If we move to 20 training samples the gradient boost algorithm's accuracy goes up to 96.6% while the NN improves to 91.1%.

By repeating the same analysis for the rest of the designs in the dataset, we could achieve accuracies over 95% using only 10 samples. In the memories, the best results came from the NN, 95% accuracy with only 300 samples. Note that we do not pick random samples from the existing dataset. We generate a new LHS and, for each new sample, finds which point in the current dataset is closer to it.

C. Predicting Core Configurations and Corners with Transfer Learning

We created a base neural network model in which we trained the default Rocket core configuration. For every new

TABLE I
MODELS EVALUATION FOR ROCKET CORE DEFAULT CONFIGURATION.

Model	Accuracy (%) = 100% - MAE					Std Dev	Relative Time / Size		
	Dynamic Pwr	Leakage Pwr	Critical Path	Area	PPA		Train	Test	Size
RBFCubic	96.26	95.10	99.06	97.48	96.97	3.63	16	7	153
RBFLinear	97.58	95.59	98.77	97.68	97.40	2.91	6	2	153
GP	91.77	91.44	95.88	91.99	92.77	7.62	726	4	76
Poly	96.74	92.80	98.40	96.40	96.09	3.31	2	4	1
MARS	97.40	94.80	99.24	97.58	97.26	3.84	47	3	1
RF	97.87	96.99	99.15	98.43	98.11	2.38	906	333	1,886
BR	97.86	96.95	99.16	98.43	98.10	2.39	924	412	1,893
AB	97.56	88.96	98.33	95.73	95.14	2.99	159	83	17
GB	97.58	96.89	99.17	98.47	98.03	2.25	217	10	73
NN	**97.59**	**96.55**	**99.11**	**97.71**	**97.74**	**2.46**	**69,009**	**1,354**	**162**
XGB	**97.84**	**97.15**	**99.21**	**98.72**	**98.23**	**1.99**	**20**	**2**	**3**
DTR	97.18	95.81	98.80	97.83	97.40	3.11	1	1	3

TABLE II
MODELS EVALUATION FOR SRAM MEMORIES.

Model	Accuracy (%) = 100% - MAE					
	RD Energy	WR Energy	Leak Pwr	Access Time	Area	PPA
RBFCubic	99.80	99.88	98.01	99.31	98.97	99.19
RBFLinear	99.82	99.89	98.26	99.39	99.05	99.29
Poly	99.59	99.47	93.83	97.56	97.10	97.51
MARS	90.34	91.69	85.82	92.37	86.88	89.42
RF	99.22	99.01	98.88	99.91	97.32	98.87
BR	99.23	99.00	98.89	99.91	97.31	98.87
AB	89.27	88.21	85.78	92.46	81.26	87.40
GB	99.48	99.36	98.54	99.48	97.98	98.97
NN	**99.55**	**99.62**	**99.27**	**99.53**	**99.20**	**99.43**
XGB	**98.53**	**98.27**	**95.96**	**98.74**	**96.39**	**97.58**
DTR	98.52	98.22	98.13	99.88	96.00	98.15

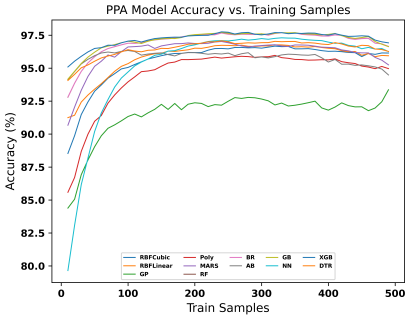

Fig. 6. Models accuracy vs. number of train samples for Rocket Default Configuration.

configuration, we are trying to predict the PPA, and we fine-tuned the output layers of this model with data from the design we are predicting. We use the same approach to predict corner operating conditions.

The neural network model used for the four layers of size $[128, 256, 512, 256, 64]$ for a total number of $313,089$ neurons. The size of this network was optimized to avoid over and underfitting. To decide the number of parameters we will share on the network, we re-trained it to predict the PPA for new core configurations for different weight sharing percentages. From Figure 7 we can see that for a low number of samples, the optimal share ratio is around 50%. This means that we will share the first three layers and re-train for each new core configuration the last two layers on our final model.

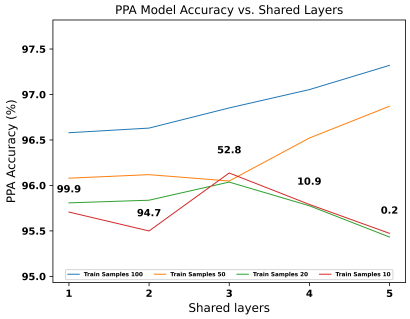

Fig. 7. Weights Shared vs Accuracy for core configuration modeling.

In Table III we can see the results for predicting the PPA for different core configurations with a base model trained with 150 samples. With this base model with only 15 training points, we can predict new core configurations PPA with 95.3% accuracy and up. We can see that component is predicted with a consistent accuracy close to the average. The leakage power component is the one with the least accuracy. One other point to consider is that if we compare with the previous PPA models we get a training speed up to $8x$.

The accuracy of the NN without transfer learning model with 20 samples for the Rocket configurations was about 91% with 20 samples. While the previous neural network structure presented in the first section of the results requires at least 40% data points to achieve 91% accuracy (see Figure 6). We can get close to 94% and up for the four configurations using only 5 samples with transfer learning. This result means that the model is learning from the previous designs.

Table IV summarizes the results of using the base model trained for $FF\_0p88V\_m40C\_Cmin$ with 50 samples. Same as before, the worst accuracy is for the leakage power due to its linearity. We can see that for 15 samples, we can predict the PPA with up to 95% accuracy for all four corners. With transfer learning, we reduce the training time up to $9x$ for

TABLE III
TRANSFER LEARNING MODELS TO PREDICT CORE CONFIGURATIONS.

Samples Train	PPA	Std Dev	Train Time Speedup
Tiny Configuration			
5	95.48	3.1	8.1
15	95.61	3.3	7.9
Small Configuration			
5	93.38	4.4	8.1
15	95.75	3.6	8.2
Medium Configuration			
5	95.02	4.0	8.7
15	95.31	3.9	6.9
Dual Configuration			
5	93.59	5.7	8.8
15	95.38	4.1	7.6

the corner conditions. Note that the area does not change for different corner conditions.

TABLE IV
TRANSFER LEARNING MODELS TO PREDICT CORNERS FOR OR1200 RESULTS.

Samples Train	PPA	Std Dev	Train Time Speedup
$SS_0p72V_m40C_Cmax$			
5	93.24	5.1	8.2
15	95.06	4.6	9.3
$SS_0p72V_125C_Cmax$			
5	91.93	6.3	7.5
15	94.23	5.5	6.8
$SS_0p72V_125C_Rcmax$			
5	92.39	5.8	9.0
15	94.14	5.4	8.4
$SS_0p72V_m40C_Cmax$			
5	91.79	6.5	7.8
15	94.05	5.7	8.0

CONCLUSIONS

This work presented a machine learning based approach to model the power, performance and area for a system-on-Chip. The proposed models can predict the PPA quickly and with high accuracy. We obtained the better results with gradient boost models and neural networks. Using different designs to test the model, we could get accuracies up to 99%. When constraining the number of samples used, we do the PPA prediction with 96% accuracy training the model with only 10 data points.

To model PPA of new designs and new operating corners conditions we used a neural network with shared weights. This transfer learning approach allows learning from previous cores and designs to estimate the PPA in new ones using a low number of data points. With only 10 to 15 new samples, we could create models to predict corner configurations and corner conditions.

ACKNOWLEDGMENT

The authors would like to thank the member companies of the CAEML IUCRC and the National Science Foundation (award CNS 16-244770) for their partial support of this work.

REFERENCES

[1] Y. Zhou, H. Ren, Y. Zhang, B. Keller, B. Khailany, and Z. Zhang, "Primal: Power inference using machine learning," in *2019 56th ACM/IEEE Design Automation Conference (DAC)*, 2019, pp. 1–6.

[2] J. L. Greathouse and G. H. Loh, "Machine learning for performance and power modeling of heterogeneous systems," in *2018 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, 2018, pp. 1–6.

[3] G. Wu, J. L. Greathouse, A. Lyashevsky, N. Jayasena, and D. Chiou, "Gpgpu performance and power estimation using machine learning," in *2015 IEEE 21st International Symposium on High Performance Computer Architecture (HPCA)*, 2015, pp. 564–576.

[4] D. Lee and A. Gerstlauer, "Learning-based, fine-grain power modeling of system-level hardware ips," *ACM Trans. Des. Autom. Electron. Syst.*, vol. 23, no. 3, Feb. 2018. [Online]. Available: https://doi.org/10.1145/3177865

[5] Y. Zhang, H. Ren, and B. Khailany, "Grannite: Graph neural network inference for transferable power estimation," in *Proceedings of the 57th ACM/EDAC/IEEE Design Automation Conference*, ser. DAC '20. IEEE Press, 2020.

[6] F. Last, M. Haeberlein, and U. Schlichtmann, "Predicting memory compiler performance outputs using feed-forward neural networks," *ACM Trans. Des. Autom. Electron. Syst.*, vol. 25, no. 5, Jul. 2020. [Online]. Available: https://doi.org/10.1145/3385262

[7] F. Last and U. Schlichtmann, "Partial sharing neural networks for multi-target regression on power and performance of embedded memories," in *Proceedings of the 2020 ACM/IEEE Workshop on Machine Learning for CAD*, ser. MLCAD '20. New York, NY, USA: Association for Computing Machinery, 2020, p. 123–128.

[8] R. Venkatesan, Y. S. Shao, M. Wang, J. Clemons, S. Dai, M. Fojtik, B. Keller, A. Klinefelter, N. Pinckney, P. Raina, Y. Zhang, B. Zimmer, W. J. Dally, J. Emer, S. W. Keckler, and B. Khailany, "Magnet: A modular accelerator generator for neural networks," in *2019 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, 2019, pp. 1–8.

[9] J. Kwon and L. P. Carloni, "Transfer learning for design-space exploration with high-level synthesis," in *Proceedings of the 2020 ACM/IEEE Workshop on Machine Learning for CAD*, ser. MLCAD '20. New York, NY, USA: Association for Computing Machinery, 2020, p. 163–168.

[10] Z. Lin, J. Zhao, S. Sinha, and W. Zhang, "Hl-pow: A learning-based power modeling framework for high-level synthesis," in *2020 25th Asia and South Pacific Design Automation Conference (ASP-DAC)*, 2020, pp. 574–580.

[11] A. Natekin and A. Knoll, "Gradient boosting machines, a tutorial," *Frontiers in neurorobotics*, vol. 7, p. 21, 12 2013.

[12] J. H. Friedman, "Greedy function approximation: A gradient boostingmachine." *The Annals of Statistics*, vol. 29, no. 5, pp. 1189 – 1232, 2001. [Online]. Available: https://doi.org/10.1214/aos/1013203451

[13] T. Chen and C. Guestrin, "XGBoost: A scalable tree boosting system," in *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ser. KDD '16. New York, NY, USA: ACM, 2016, pp. 785–794.

[14] Q. Wang, Y. Ma, K. Zhao, and Y. Tian, "A comprehensive survey of loss functions in machine learning," *Annals of Data Science*, 04 2020.

[15] D. Eriksson, D. Bindel, and C. A. Shoemaker, "pysot and poap: An event-driven asynchronous framework for surrogate optimization," *arXiv preprint arXiv:1908.00420*, 2019.

[16] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg *et al.*, "Scikit-learn: Machine learning in python," *Journal of machine learning research*, vol. 12, no. Oct, pp. 2825–2830, 2011.

[17] M. D. Shields and J. Zhang, "The generalization of latin hypercube sampling," *Reliability Engineering and System Safety*, vol. 148, pp. 96–108, 2016.