SPEED: Switching Power Estimation using Machine Learning with Time-Efficient Data Collection*

Hoang Nguyen
Electrical Engineering and Computer Science
University of California, Berkeley
Berkeley, CA, USA
hoang_nguyen@berkeley.edu

Yuki Ito
Computer Science
University of California, Berkeley
Berkeley, CA, USA
itoyuki@berkeley.edu

***Abstract*—Accurately estimating the switching power in VLSI circuits is crucial for achieving high performance design, but pushing a circuit through the VLSI design flow from synthesis, Place-And-Route (PAR), gate-level simulation, and gate-level analysis for can take hours to days depending on the scale of the design. Previously, Machine Learning (ML) models such as Graph Neural Network (GNN), Convolutional Neural Network (CNN), and regression models were shown to have the ability to infer the toggle rates with a speed up of 10 to 20 times relative to running PAR. Although these models show great speed-ups, the training data were obtained from post-PAR gate-level power analysis, which is still time-consuming for data collection. Here we show developed Switching Power Estimators that are linear regression-based, trees and ensemble learning-based, and neural network-based with data collected from RTL-level simulation and power analysis. These estimators were validated on a few representative combinational logic circuits and their performance were measured using the NRMSE metric. We believe that the exploration of such Switching Power Estimators is crucial for designs that require instant power estimation from data collection to prediction making, and can be extended for future research.**

***Index Terms*—Switching Power Estimation, VLSI, Data Collection, Featurization, Machine Learning, Linear Regression, Decision Trees and Ensemble Learning, Neural Network.**

I. Introduction

Accurate power estimation is a critical part for setting appropriate thermal and electrical constraints to achieve state-of-the-art performance on a VLSI design. Power dissipation found in VLSI designs can be described as

$$P_{total} = P_{dynamic} + P_{static} \tag{1}$$

where P_{total} is the total power consumption of the circuit, $P_{dynamic}$ and P_{static} is the total dynamic power and static power, respectively. Static power is comprised of leakage, or current that flows through the transistor when there is no activity. Dynamic power is comprised of switching and short-circuit power.

$$P_{dynamic} = P_{switching} + P_{short-circuit} \tag{2}$$

where $P_{switching}$ is the total power dissipated during switching/toggling activities and $P_{short-circuit}$ is the total power drawn due to short-circuit current. The switching power can be modeled as

$$P_{switching} = \alpha C V_{DD}^2 f \tag{3}$$

where α is the switching activity factor of the circuit, C is the load capacitance that the circuit is driving, V_{DD} is the voltage supplied to the circuit by the power rail, and f is the operating clock frequency of the circuit. Knowing switching power of a given circuit can give insight to its performance - allowing us to make further power optimizations when integrated with our subcircuits in a design.

In order to estimate the average power over a large number, for instance, millions, of simulation cycles, ones would need to perform architectural power analysis which consume hours to days depending on the switching activity factors of the logic gates and the scale of the design. An alternative proposed to estimating power or estimating the switching activity factor is performing switching activity estimation instead of calculating the average switching factors directly for computing the dynamic power. A Switching Activity Estimator (SAE) uses a probabilistic approach to estimate average toggle rates for unit inputs and registers in a simulation, allowing us to completely cut out the process measuring power cycle-by-cycle. Having said that, given their probabilistic nature, SAEs often run into issues regarding accuracy, signal correlation or reconvergence. There have been numerous attempts for a over-the-year development of high-accuracy SAE to cut computation and memory cost.

Recent trend of Machine Learning (ML) approaches to creating an SAE have shown drastic improvement of accuracy and reduction of computational overhead. These work explored usages of various ML models for predicting the switching activities such as GRANNITE with Graph Neural Network (GNN) [1], APOLLO with Minimax Concave Penalty (MCP)-based feature selection [3], PRIMAL with Convolutional Neural Networks (CNN) [2], Simmani with Clustering [4], and GPGPU with Neural Networks [5]. However, these implementations do not take into account the time it takes to collect the data to train these ML methods. Many ML methods, especially Neural Networks, require thousands of stimulus and power traces to successfully train with feasible accuracy.

* This project is overlapped with the design project for the CS 289A course taught by Professor Jonathan Shewchuk that we are taking this semester.

Training these abovementioned ML methods on RTL-level power dataset have not been explored to analyze the tradeoff of not using gate-level power dataset. In addition, some flavors of regression-based methods such as LASSO and ensemble learning methods such as Random Forests and Decision Trees with AdaBoost have not been explored extensively.

In addition, many of GNN/CNN topologies are often treated as a black box to achieve high accuracy in estimating switching activities and therefore lack model interpretability. Being able to infer the feature weights that a ML model has learned is essential in understanding and interpreting why certain circuits expose some trend in power with varying input.

In this project, we contribute to this effort by exploring the estimation of switching power using three ML-based methods, namely, linear regression, decision trees and its variants with ensemble learning, and neural network.

II. Background and Previous Work

A. GRANNITE

GRANNITE [1] is a GPU-accelerated novel graph neural network (GNN) model for transferable vector-based average power estimation. It is a supervised learning-based SAE that foregoes the need for gate-level simulation and can be all done in RTL simulation. They utilized register states and unit inputs from RTL simulation as features and infer the combinational gate toggle rates. Transferability grants GRANNITE the ability to estimate combinational gate toggle rates across any design without the need for retraining. By translating gate-level netlists to graph objects in a one-to-one mapping while encoding sequential hierarchy within the graphs, they embed information such as pin state, intrinsic state probabilities, intrinsic transition probabilities, and etc. They tested this GNN estimators across numerous benchmark circuits such as a 32-bit fixed point adder, 32-bit floating point multiplier, and RISC-V Rocket Core (SmallCore), and found that GRANNITE achieved a speedup of more than 18.7 times with only less than 5.5% error across these circuits. As time efficiency is a key point to consider for power analysis and estimation, such significant speed-up is novel alongside its transferability.

B. PRIMAL

PRIMAL [2] is a novel learning-based framework for fast and accurate cycle-to-cycle power estimation in the ASIC design flow. Training were done on numerous ML models such as ridge regression, gradient tree boosting, Multi-Layer Perceptron (MLP), and Convolutional Neural Network (CNN). They were able to achieve an average error of less than 1% across realistic benchmarks with 15X speedup. The PRIMAL design flow consists of generating RTL simulation traces containing information on RTL register and I/O signal switching activities. These traces were used as input features for the numerous ML models they trained on. For their CNN implementation, rather than directly treating its input of some circuits as a feature, they performed concise encoding to embed features. Concise encoding, which refers to switch encoding is the utilizing signal switching activities as features rather than its raw inputs for a particular design. In performing ridge regression, they also utilized Principal Component Analysis (PCA) for feature optimization as a preprocessing technique. For their CNN implementation, they mapped each RTL register into a pixel of the CNN layer after performing *node2vec* for node embedding. Similar to GRANNITE [1], they had a diverse set of benchmark designs such as a 32-bit floating pointer adder, multiplier, Network-on-chip router for CNN accelerator, and a RISC-V Rocket Core to test their accuracy on numerous ML models. As a result, their CNN model served as the most accurate across these benchmark designs with long training times (roughly 20 hours for a RISC-V Rocket Core).

C. APOLLO

APOLLO [3] is the first runtime on-chip power meter (OPM) that achieves per-cycle resolution and $<1\%$ area/power overhead. APOLLO contains a piece of hardware that is implemented on top of a design to allow per-cycle power calculations (a.k.a runtime power estimation). Unlike previous works on OPMs, APOLLO achieves both low overhead and also automated power estimation framework. APOLLO uses the minimax concave penalty (MCP)-based feature selection algorithm to automatically select less than 0.05% of RTL signals as power proxies - achieving $R^2 > 0.95$ on Arm Neoverse N1 and $R^2 > 0.94$ on Arm Cortex-A77 microprocessors. Usually to test power proxy selection method for an OPM, one provide 1) random stimuli, 2) realistic workloads, 3) handcrafted ISA tests or micro-benchmarks. But hitting all edge cases for the microarchitecture specific benchmarks is very difficult task. Therefore, APOLLO auto-generated the training set of micro-benchmarks using Genetic Algorithm (GA) to train the MCP-based feature selection algorithm.

D. Simmani

Simmani [4] is a framework that automatically selects signals most correlated with power dissipation and trains power models in terms of the selected signals in RTL. Clustering signals that show similar toggle patterns allows to see distinctive signal patterns that we can characterize the power. With the selected signal, they trained power models using cycle-accurate power traces to reduce computational overhead. In order to cluster signals to view the toggle patterns, they constructed a toggle-pattern matrix, which represents an RTL signal by time window of the simulation. The toggle-pattern matrix is constructed from the VCD dumps for the clustering problem, where we can observe tendencies of RTL signals by computing the Singular-Value Decomposition (SVD) of the matrix and performing k-means++ clustering. After selection, Simmani performs an elastic net regression (combination of LASSO and ridge) on the selected signals with per-cycle power traces.

E. GPGPU

GPGPU [5] is a GPU power estimation model where its accuracies are within 15% to 10% of cycle-level simulators.

GPGPU performs cluster classification on certain performance trends a GPU oversees within a simulation. The classification then outputs a kernel that characterizes the power in relation to frequency and numerous other parameters that can be used as a Look-Up Table (LUT). The clustering process is done by k-means clustering and the classifier is created via a Neural Network.

III. Methodology

Our project consists of two main tasks: 1) collecting and featurizing the training and validation data from RTL-level simulation and power analysis, and 2) design, train, and evaluate the ML models against the obtained input data and labels.

A. RTL Simulation and Power Analysis, Data Collection and Featurization

Figure 1 shows the three main categories of combinational logic circuits. To prove that predicting the switching power of any combinational logic circuits is achievable, we chose 1-2 representative combinational logic circuits from each category to collect data and perform power estimation on. For the arithmetic and logical functions type, we chose the ALU, which covers addition, subtraction, comparison, etc. Specifically, we used a 32-bit 2-input ALU (Figure 2a) and a 64-bit 2-input ALU (Figure 2b) in order to check whether different number of bits affect the performance of our machine learning model. Additionally, we chose to collect data for a 32-bit 2-input multiplexer (MUX) (Figure 2c) to cover the data transmission combinational circuit type, and a BCD to 7-segment decoder for 32-bit integers (Figure 2d) to represent the code converters category.

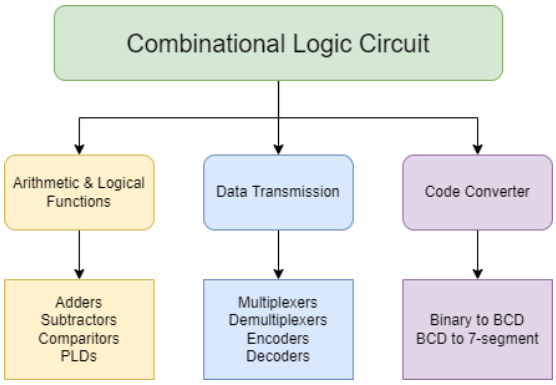

Fig. 1: Combinational logic-type circuits can be divided into three categories: arithmetic and logical functions, data transmission, and code converters. This figure is acquired from Electronics Tutorials [9]

1) RTL-level Simulation and Power Analysis: We pushed the abovementioned combinational logic Devices-Under-Test (DUTs) through the RTL-level Simulation and Power Analysis flow by utilizing the HAMMER tool.The HAMMER design flow allows smooth, portable VLSI design process to invoke

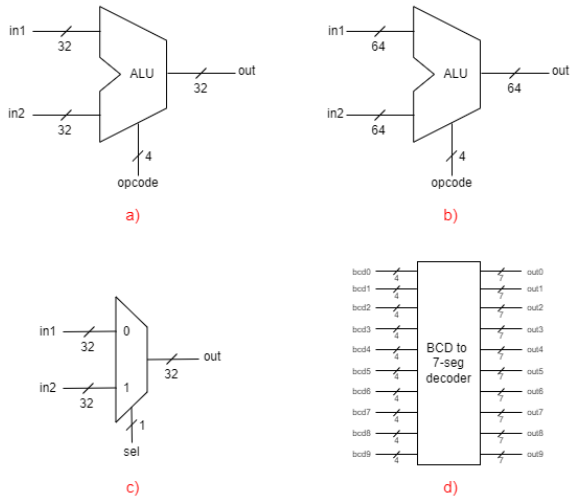

Fig. 2: Representative combinational logic circuits that we used for switching power estimation.

RTL-level simulation and power analysis by consuming configuration files in form of YAML and JSON. For the RTL-level simulation tool, we used Synopsys VCS, a functional verification, simulation, and constraint solver engines. Within our RTL-level simulation, we set our clock period to 1.0 ns and timescale of 1 ns broken up into 10 ps precision. Because we are simulating combinational logic circuits, we don't require to clock our input pins with some constrained clock frequency, but just enough to propagate the signals back into the outputs.

After RTL-level simulation, we invoked Cadence Joules and passed in the RTL simulation traces. We provided Joules with the transistor technology to construct the digital circuits (ASAP7 technology) and paths to the lib files containing the Process-Voltage-Temperature (PVT) corners of the technology in a TCL file. These information are the foundation that generates the overall power consumption of our design. After loading the stimuli files from our RTL simulation traces, Joules perform their own method of clock tree synthesis and output a power profile of the simulation.

2) Data Collection: In this project, we needed to generate our own training and validation switching activity and power data. Even though such data exist, they are proprietary for commercial purposes, and are unavailable to the public. Since we spent most of our time on data collection, we intended to create a pipeline that other researchers can potentially reuse to collect training and validation data on their design of choice. Figure 3 illustrates our data collection pipeline, which consists of a Verilog Generation Module who generates test vectors and consumes the RTL module and testbench templates that we prepared to generate the actual Verilog files in the .v format. Next, the Simulation and Data Collection Module consumes the generated Verilog files, along with the config and automation files in YAML, SDC, and TCL formats for invoking Synopsis VCS to perform RTL-level simulation as well as Cadence Joules for RTL-level power analysis, and

collect the switching activities of each input bit of the DUT as the training samples and its switching power as the grouth truth labels to train against. Lastly, the collected data is compiled into a CSV-format file, consumed by the Preprocessing and Featurization Module, and converted into a .mat format that can easily be loaded for machine learning purposes.

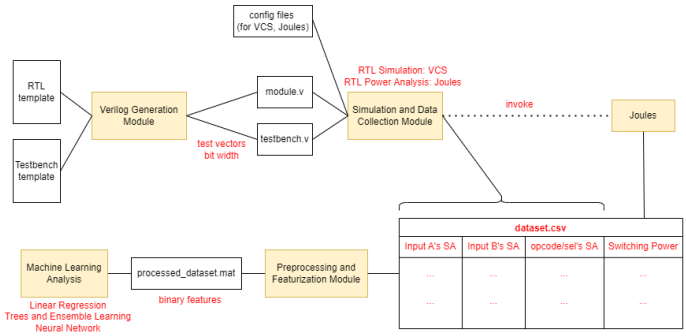

Fig. 3: SPEED's data collection pipeline. Additional modification can be done in future work to support data collection on arbitrary Verilog DUTs.

3) Featurization: For machine learning purposes, it is important to identify and engineer features that are useful when training the machine learning models. In this project, we featurized our training data using the switching activities in the input gates, including operand bits as well as *opcode* bits for the ALU circuits and the *sel* (select) bit for the MUX circuit. To create our binary features for our training samples, we encode a 0 if the state of a bit/input gate remains the same, and encode a 1 if that bit/input gate switches state during the simulation. For instance, the 32-bit and 64-bit 2-input ALU has 64 and 128 features representing whether or not the operand bits toggled, respectively, and 4 features representing the toggle state of the 4-bit opcode inputs. Similarly, the 32-bit 2-input MUX has 64 features and 1 additional feature representing the operand bits' toggle state and the *sel* bit's toggle state, respectively. Lastly, the BCD to 7-segment decoder for a 32-bit integer requires 40 features, with 4 features representing the toggle state of each input ports. This is because the largest 32-bit integer is $2^{32} - 1$ which has 10 digits in decimal base, so the hardware needs to have 10 4-bit BCD inputs (and hence 10 7-bit 7-segment output). The summary of the number of features for each circuit can be seen on Table I.

TABLE I: Number of encoded features for each combinational circuit used in this project.

Circuit	Number of Features
32-bit 2-input ALU	68
64-bit 2-input ALU	132
32-bit 2-input MUX	66
BCD to 7-segment Decoder	40

B. Machine Learning Models Creation, Training, and Evaluation

During data collection, we acquired approximately 5000 samples for each of the four abovementioned combinational logic circuits. We performed a 80/20 split for training and validation, so roughly 4000 samples were used for training and the other 1000 were used for validation. The machine learning methods we chose to employ for estimating the switching power on our combinational logic circuits are 1) linear regression, 2) trees and ensemble learning, and 3) neural network.

1) Linear Regression: For linear regression, we explored least squares linear regression, ridge regression, and LASSO, which can be easily implemented from scratch or using scikit-learn. Least squares linear regression fits a high-dimensional hyperplane to high-dimensional sample points. Training a least square regression model requires solving the following optimization problem:

$$w^* = argmin_w ||X_t w - y_t||_2^2 \tag{4}$$

where w^* is the trained weight, X_t and y_t are the training data and training labels respectively.

Ridge regression [6] is similar to least squares linear regression, but with an addition of an L2 regularization term. Ridge regression provides a unique solution to the least squares optimization problem and penalizes large weights to reduce variance, a source of validation error of the estimator, which in turn reduce overfitting. LASSO [7] is a linear regression method that has embedded feature subset selection. LASSO is also similar to least squares linear regression, but with an addition of an L1 regularization term, which encourages sparsity and helps eliminate useless features, thus reduces variance of the estimator and reduces overfitting.

After training (i.e. solving the normal equations of the above optimization problems), the optimal set of weights w^* can then be used to predict the labels (switching power) in the validation set. The predicted label vector can be computed by

$$\hat{y_v} = X_v w^* \tag{5}$$

where $\hat{y_v}$ is the predicted label vector containing the predicted switching power for all collected validation data and X_v is the validation data.

2) Decision Trees, Random Forests, and Boosted Decision Trees: Decision tree is a nonlinear method for classification and regression. It is most effective when solving machine learning problems with the features and labels follow a nonlinear boundary, which linear regression, even with feature engineering, fails to perform. In the naive decision tree that we implemented, we performed slits on every feature.

Each node of a decision tree represents the feature being split at and its overall loss contribution. Decision trees split on the feature that maximize the information gain. However, because decision trees learn a nonlinear boundary, it can especially be susceptible to overfitting. An implementation of decision trees where its depth is high enough to give us 100% training accuracy does not guarantee the same reflection upon validation.

To combat overfitting, we can use ensemble learning. Random forests is a method of ensemble learning, which trains

numerous deep decision trees with high variance (i.e. proned to overfit). The overall variance can be reduced by averaging the predictions of all of decision tree learners.

AdaBoost (Adaptive Boosting) is another strain of ensemble learning like random forests with the exception that it tries to adapt and learn from previous accuracy of learners. AdaBoosted Decision Trees work by training and measuring the performance of estimators one by one and construct a new predictor based on the knowledge of the previous predictors' performances. The AdaBoosted metalearner takes a weighted average/vote on all constructed estimators. Decision trees, random forests, and AdaBoosed trees were implemented using scikit-learn.

3) Neural Network: Neural network is a universal function approximator, and with appropriate hyperparameters and topologies, can potential solve a vast number of machine learning problems. In the domain of neural network, we investigated Multi-Layer Perceptrons (MLPs) with various hidden layers and whether or not we include a dropout layer. Figure 4 shows an example neural network architecture. The number of input neurons is the number of features shown in Table I, the number of hidden neurons were fixed at 1024 for all hidden layers for simplicity, and there is only one output neuron since switching power estimation is a regression problem. Dropout is a method to reduce overfitting in a neural network-based estimator. In our experimental setup, the dropout layer, if exists, is always applied right before the final Fully Connected (FC) layer and the dropout rate is fixed at 0.1.

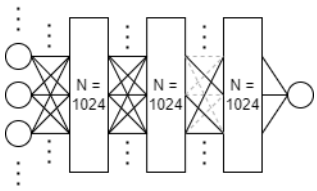

Fig. 4: An illustration of an example neural network architecture with 3 hidden layers and a dropout layer.

We designed 7 different architectures of neural network, including topologies with 0, 1, 2, and 3 hidden layers without dropout, and topologies with 1, 2, and 3 hidden layers with dropout. The neural network models were implemented and trained using Pytorch.

C. Evaluation and Comparison Method

Our chosen metric of measuring the training and validation error is the Normalized Root Mean square Error (NRMSE). The NRMSE can be calculated as:

$$NRMSE = \frac{1}{y_{max} - y_{min}} \sqrt{\frac{\sum_{i=1}^{n} (y_i - \hat{y}_i)^2}{n}} \qquad (6)$$

where y_{max} and y_{min} are the largest and smallest values of the switching activity factor label vector, respectively, y_i is the switching activity factor label for the i-th sample, $\hat{y}_i$ is the predicted switching activity factor of the i-th sample, and n is the total number of samples.

This error metric is almost the same as the widely used Root-Mean-Square Error (RMSE), with the exception that we normalized it by dividing the RMSE by the difference from the maximum label to the minimum label [8]. This allows us to reasonably compare the performance of different machine learning methods on a particular circuit, as well as comparing the performance of a particular machine learning model on various circuits.

IV. Results and Discussion

We trained the machine learning-based switching power estimators with the hyperparameters shown in Table II. These hyperparameters were acquired from hyperparameter tuning and hyperparameter values with the lowest validation NRMSE were selected. For linear regression-based estimator, we used least squares model which surpassed ridge regression and LASSO in performance, measured in validation NRMSE. For the random forest-based estimator, training takes a large amount of time since training a random forest is equivalent to training an ensemble of decision tree estimators. Due to the computational resource limitation and time limitation of a semester long project, we fixed the number of trees (i.e. estimators) in an ensemble to be at 100 trees (i.e. estimators).

TABLE II: Hyperparameter values that achieve the best validation performance from hyperparameter tuning. The tuned hyperparameters include the tree depths for decision trees with AdaBoost and the neural network architecture. the dropout probability p denotes how many connections in a FC layer to be dropped. A low p value means fewer connections were being dropped, while a high p value means more connections were being droped.

Circuit	Best Tree Depth (Boosted Decision Trees)	Architecture (Neural Net)
32-bit ALU	7	3 hidden layers, no dropout
64-bit ALU	4	2 hidden layers, dropout with p = 0.1
32-bit MUX	5	3 hidden layers, dropout with p = 0.1
BCD-7seg decoder	6	3 hidden layers, dropout with p = 0.1

Figure 5 shows the validation error in terms of NRMSE for each switching power estimation methods on the representative combinational logic circuits. It is interesting to note that the 32-bit 2-input ALU, the 32-bit 2-input MUX, and the BCD to 7-segment decoder for a 32-bit intergers sees the linear regression-based and neural network-based estimators to perform better than the trees and ensemble learning-based estimator. However, on the 64-bit 2-input ALU, the trees and ensemble learning methods for the switching activity estimator perform better than the linear regression- and neural network-based methods. This results correlate with our hypothesis that the performance of the machine learning-based switching power estimator changes across different circuits with different number of input bits. Looking at Table I, we can see that the number of features for the 32-bit circuits cluster in the range

of 40 to 68 features, where as the 64-bit ALU circuit has roughly twice as much the number of features. This causes the 64-bit ALU circuit to be the most proned to overfitting, and explains why linear regression and neural network methods without ways to reduce variance to perform worse, whereas the ensemble learning methods reduces the bias and variance of the estimators achieved lower validation error. On the other hand, the 32-bit circuits are more likely to underfit, so methods such as random forest to average out various decision tree estimators and reduce the variance are not as useful.

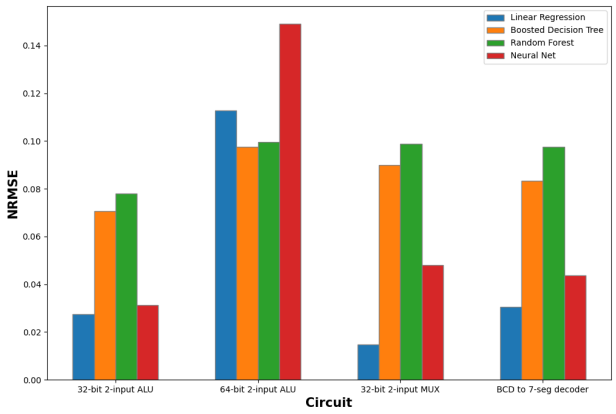

Fig. 5: Comparison of the performance of linear regression, boosted decision tree, random forest, and neural network candidates for the switching activity estimator on predicting the switching power on 4 representative combinational logic circuits.

For the 32-bit circuits, the linear regression-based switching power estimator achieves the best performance. Specifically, the it received a NRMSE of 0.02747 on a 32-bit 2-input ALU, 0.01464 on a 32-bit 2-input MUX, and 0.03052 on a BCD to 7-segment decoder for 32-bit integers. For the 64-bit 2-input ALU, decision trees with AdaBoost achieved the best performance with the NRMSE of 0.09751.

V. Future Work

In this project, we only featurized the switching activities from input gates. However, we believe it would be useful to also featurize the switching activities from intermediate and output gates. This would allow our data to be lifted to a higher dimension, and have more features to potentially fit the training/validatidation data better (i.e reduce the bias of the estimator). With that said, we would need to incorporate some feature subset selection or dimensionality reduction scheme to remove unhelpful features, since purely adding features/dimensions would increase the variance of the estimators which causes overfitting. This is a good segway to our next item for future work, which is investigating further machine learning method to be used for the switching activity estimator. Methods like running PCA before linear regression to eliminate useless features could be explore.

Lastly, our budget of time this semester only allows us to explore the feasibility of switching power estimation on combinational logic circuits, but most digital circuits nowadays have sequential circuitry in their datapath. Therefore, further develop to the data collection pipeline to collecting switching power data on sequential circuits would be necessary.

VI. Conclusion

In conclusion, we have built and trained three types of switching activity estimators, namely linear regression, decision trees and ensemble learning, and neural network, and evaluated the estimators using the NRMSE metric. We have also generated switching power datasets, where input features denote whether the input gates switches state, and ground truth labels are the switching power computed by Cadence Joules, on representative combinational logic circuits, including 32-bit and 64-bit 2-input ALUs, a 32-bit 2-input MUX, and a BCD to 7-segment decoder for 32-bit integers. Our data collection pipeline, with additional enhancements, could potentially be used for acquiring training data for the switching power estimators on more complex digital circuits.

References

[1] Y. Zhang, H. Ren, and B. Khailany, “GRANNITE: Graph Neural Network Inference for Transferable Power Estimation,” in 2020 57th ACM/IEEE Design Automation Conference (DAC), Jul. 2020, pp. 1–6. doi: 10.1109/DAC18072.2020.9218643.

[2] Yuan Zhou, Haoxing Ren, Yanqing Zhang, Ben Keller, Brucek Khailany, and Zhiru Zhang. 2019. ”PRIMAL: Power Inference using Machine Learning.” In Proceedings of the 56th Annual Design Automation Conference 2019 (DAC ’19). Association for Computing Machinery, New York, NY, USA, Article 39, 1–6. doi: https://doi.org/10.1145/3316781.3317884

[3] Z. Xie et al., “APOLLO: An Automated Power Modeling Framework for Runtime Power Introspection in High-Volume Commercial Microprocessors,” in MICRO-54: 54th Annual IEEE/ACM International Symposium on Microarchitecture, New York, NY, USA, Oct. 2021, pp. 1–14. doi: 10.1145/3466752.3480064.

[4] Donggyu Kim, Jerry Zhao, Jonathan Bachrach, and Krste Asanović. 2019. ”Simmani: Runtime Power Modeling for Arbitrary RTL with Automatic Signal Selection.” In Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO ’52). Association for Computing Machinery, New York, NY, USA, 1050–1062. doi: https://doi.org/10.1145/3352460.3358322

[5] 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), Feb. 2015, pp. 564–576. doi: 10.1109/HPCA.2015.7056063.

[6] A. E. Hoerl and R. W. Kennard, “Ridge Regression: Biased Estimation for Nonorthogonal Problems,” Technometrics, vol. 42, no. 1, pp. 80–86, 2000, doi: 10.2307/1271436.

[7] R. Tibshirani, “Regression Shrinkage and Selection Via the Lasso,” Journal of the Royal Statistical Society: Series B (Methodological), vol. 58, no. 1, pp. 267–288, 1996, doi: 10.1111/j.2517-6161.1996.tb02080.x.

[8] “How to normalize the RMSE.” https://www.marinedatascience.co/blog/2019/01/07/normalizing-the-rmse

[9] “Combinational Logic Circuits using Logic Gates,” Basic Electronics Tutorials, Aug. 01, 2013. https://www.electronics-tutorials.ws/combination/comb_1.html.