Explorations in Application of Machine Learning in Power Estimation of Digital Systems Pre-Synthesis

Kevin Hoser[1], Addison Raak[1] and Nazanin Mansouri[1#]

[1]Shiley School of Engineering, University of Portland
[#]Advisor

ABSTRACT

This paper presents an approach to exploit Machine Learning (ML) to accurately and efficiently predict power consumption in digital systems at the higher-levels of abstraction prior to synthesis. As valuable resources and much time are invested when developing new products, designers can greatly benefit to know early in the design process if the final design's power consumption is within the reasonable margins of the given constraints. This is done by analyzing the high-level models of the design (behavioral or register transfer level) and without investing resources for synthesizing the design. We have used machine learning models trained on tallies of cell groups that were parsed from gate-level netlists in order to estimate the design's "internal", "switching", "leakage", and "total" powers. Four supervised learning models, *Multi-Layer Perceptron* (MLP), *Ridge Regression*, *Elastic Net*, and *K-Nearest Neighbors*, were evaluated across three different technologies: 90 nm, 45 nm, and 15 nm cell libraries. Our experiments provide a meaningful comparison of these models for the 3 technology nodes. The most successful model in the 15 nm library was MLP, which had the smallest error in predicting total power. Additionally, MLP models improved the average error when predicting a single power component (internal, switching, or leakage), compared to simultaneously predicting all three power components in a single model.

Introduction

Power consumption remains a main challenge in the design of digital electronics. Every new generation of microelectronic products needs to meet more strict power requirements to remain competitive in the market. Power management has become more crucial with the rise of data centers, Internet of Things (IoT) devices, and personal smart devices. Consumers desire long lasting batteries and low power bills, so designers attempt to keep power consumption low. At the same time, in architectures below 100 nm transistors do not behave as perfect switches and leakage power drastically increases. The scaling issue only worsens as designers incorporate sub-7nm technology. Additionally, due to an increase in complexity, ultra large-scale integration (ULSI) designs can take large amount of computing time to optimize critical constraints such as power. Consequently, power has come to the forefront of the issues. It has also become more critical than ever to predict a design's power usage early in the design cycle and at the higher levels of abstraction.

High-level synthesis (HLS) is the process of automatically synthesizing a design at the register transfer level of abstraction (RTL) from a behavioral description that models its desired behavior. *Logic Synthesis* that follows HLS, takes the design to a lower level of abstraction. Logic Synthesis is the process of automatically synthesizing a design at the gate level of abstraction from an RTL design. The HLS optimization process relies on estimating a

Style Definition: jsrAuthor
Style Definition: jsrH1
Style Definition: jsrH2
Style Definition: jsrPara
Formatted: Font: NimbusSanL
Formatted: Font: NimbusSanL, Not Bold
Formatted: jsrTitle, Left
Formatted
Formatted: Font: NimbusSanL
Formatted: Font: NimbusSanL, Not Bold
Formatted: Font: NimbusRomNo9L, 10 pt
Formatted: jsrPara
Formatted: Font: NimbusSanL
Formatted: Font: NimbusRomNo9L, 10 pt
Formatted: Font: NimbusSanL, 10 pt
Formatted: Font: NimbusRomNo9L, 10 pt
Formatted: Font: NimbusSanL
Formatted: Font: NimbusRomNo9L, 10 pt, Not Bold
Formatted: jsrPara
Formatted: Font: NimbusRomNo9L
Formatted: Justified
Formatted: Font: NimbusRomNo9L, 10 pt, Font color: Black
Formatted: jsrPara
Formatted: Font: NimbusRomNo9L
Formatted: Justified
Formatted: Font: NimbusRomNo9L, 10 pt
Formatted: jsrPara
Formatted: Font: NimbusRomNo9L
Formatted: Justified

configuration's performance to achieve the most optimal design [1]. Therefore, HLS tools can benefit from efficient estimation techniques to quickly predict a design's power consumption.

In recent years, machine learning has gained popularity in the electronic design automation (EDA) field. For instance, it has been proven successful in selecting design constrains in logic synthesis and physical design [2], in pre-routing timing prediction [3], in predicting formal verification resources [4], or in predicting the cost for moving data between memory/storage units [5]. These experiments make machine learning appear as a promising tool to solve synthesis problems or to optimize synthesis flows. Machine learning determines properties and trends from a large set of ~~samples~~sample (training) data to infer the same type of properties and trends for a new set of data, and it is highly versatile in its application. In addition, machine learning is accessible from open-source packages and has a relatively low computing cost once a model has been configured. However, new ways are still being discovered to integrate machine learning into high-level design and synthesis processes.

In what follows, we explore how machine learning can be employed to predict a design's power consumption early in the design cycle and at the higher levels of abstraction. In our experiments we used small to medium size designs from a library of behavioral Verilog models (behavioral descriptions expressed in Verilog hardware description language) during the training process. The features of each behavioral description are extracted, then the design is synthesized, and finally the power usage of the resulting synthesized netlist is measured and recorded. Given a new behavioral description, our machine learning models can extract its features and predict the power consumption of the final synthesized designs with reasonable accuracy. This helps eliminate design solutions that would not meet the power constraints without investing time and resources in synthesis processes that are certain to fail due to strict power budgets.

We used four different machine learning models and three different technology libraries. Design features were extracted from several gate-level netlists synthesized from a relatively large and diverse set of benchmarks. In addition, the power consumption of each design was computed by PrimeTime [6], the timing and power analysis tool from the Synopsys Suite of design tools [7]. The design features and power consumption values of a large subset of high-level designs were provided to the ML models during the learning phase, while the designs in the remaining smaller subset (not seen by the ML models) were used to evaluate the accuracy of these models in predicting power consumption of the designs after the learning phase. To have a meaningful comparison, the exact same training and testing design sets were used across all four models. Our experiments confirmed that ML can be effectively employed for predicting power consumption of synthesized digital systems.

This approach is discussed in depth in the following sections. The organization of the paper is as follows: (1) an explanation on the significance of this research, (2) a survey of related research, (3) a detailed procedure of our methodology, (4) a summary of our findings, and (5) an elaborate discussion of our findings.

Motivation

Predicting power consumption accurately has great value, and can be applied to optimize processes in different stages in the design cycle. The significant added benefit and the contribution of a machine learning approach to power estimation is on developing a knowledge-base that directly relates design features to power cost. Each generation of designs learn from previous design experiences, and contributes to the future ones, hence, continuously enhancing the knowledge and refining the accuracy of predictions.

~~2~~

Formatted: Font: NimbusRomNo9L
Formatted: Font: NimbusRomNo9L
Formatted: Font: NimbusRomNo9L
Formatted: Font: NimbusRomNo9L
Formatted: Font: NimbusRomNo9L
Formatted: Font: NimbusRomNo9L
Formatted: Font: NimbusRomNo9L
Formatted: Font: NimbusRomNo9L
Formatted: Font: NimbusRomNo9L
Formatted: Font: NimbusRomNo9L
Formatted: Font: NimbusRomNo9L
Formatted: Font: NimbusRomNo9L
Formatted: Font: NimbusRomNo9L
Formatted: jsrPara, Indent: First line: 0"
Formatted: Justified
Formatted: Font: NimbusRomNo9L, 10 pt
Formatted: jsrPara
Formatted: Font: NimbusRomNo9L
Formatted: Justified
Formatted: Font: NimbusRomNo9L, Font color: Black
Formatted: jsrPara

1. In many development groups the front-end and back-end work are done independently. Synthesis begins in the back-end only when the design work in the front-end is near complete. Learning that the design implementation fails to meet its power budget after synthesis is too costly. At this point, it is too late and too hard to change high-level design decisions that impact power. Such a failure will directly affect the time-to-market of the product. As a contribution of the approach presented in this paper, the power analysis knowledge from back-end can be brought into front-end and be used for guiding design decisions.

   Using our ML approach, throughout the front-end development phase, designers can directly benefit from the previous learnings to actively evaluate the power cost of their design decisions. For example, a designer can learn from the knowledge-base to flag early on that certain design constructs or certain design decisions can lead to high power consumption in synthesized hardware. These constructs or decisions can then be avoided and prevent wasting precious time invested in synthesis and the need to iterate the design cycle when constraints are not met.

2. The ML power-estimation engine can also be used as part of the cost function of a high-level synthesis tool to guide design space exploration and aid in synthesis decisions that lead to power optimal designs.

Related Research

Several researchers have predicted the power consumption of digital designs by examining the relationship:

$$P_{dyanmic} = \frac{1}{2} C_{eff} \times {V_{dd}}^2 \times F \times A$$

where C_{eff} is the effective capacitance, V_{dd} the voltage, F the clock frequency, and A the switching activity. Commonly, this analysis has taken place at the register transfer level (RTL) to investigate the relation of switching activity and capacitance, otherwise known as switching capacitance, on power consumption [8]- [9]. Buyuksahin and Najm proposed a methodology that estimated power by taking the product of the estimated total capacitance and the estimated average switching activity in [8]. Capacitance was modeled with respect to area, by multiplying the average capacitance in a cell library with the design's total number of cells. Their switching activity model expanded on a previous estimator that used the design's number of inputs and outputs as its parameters.

Yang et al. discussed power estimation trade-offs between accuracy and simulation time in [9]. Most significantly, they noted that netlist offers a high accuracy and ease of extraction from high level designs. Only post-silicon methods have a higher accuracy in estimating power. However, they are impractical due the late access of post-silicon measurements in the design flow. Furthermore, they introduced a power estimator, which implemented machine learning, to overcome the challenge of identifying power trends in complex designs. Their model observed signal traces of critical registers in order to predict power.

Most recently, Zhou et al. [10] proposed a machine learning power estimator that trained on signal and power traces, derived from RTL specifications, gate-level netlists, and testbenches. Unlike others, they utilized a wide range of machine learning models, such as ridge regression, multi-layer perceptron (MLP), and convolution neural network (CNN). The authors found their greatest accuracy using non-linear models, especially CNNs. CNNs expand MLP models as they regress matrices into smaller shapes, in order to fix input of an MLP model. Like these authors, CNNs tend to have images as their inputs. Moreover, Zhou et al. argued that CNN are more scalable than MLP because the number of parameters remains constant as the input image scales up. Therefore, CNNs may prove successful to predict

Formatted: Font: NimbusRomNo9L
Formatted: Justified
Formatted: Justified, Indent: First line: 0.42"
Formatted: jsrPara, Indent: Left: 0"
Formatted: Justified
Formatted: jsrPara, Tab stops: Not at 0.64" + 1.27" + 1.91" + 2.54" + 3.18" + 3.82" + 4.45" + 5.09" + 5.73" + 6.36" + 7" + 7.63" + 8.27" + 8.91" + 9.54" + 10.18"
Formatted: Font: NimbusRomNo9L, 10 pt
Formatted: Font: NimbusRomNo9L
Formatted: jsrPara
Formatted: Font: NimbusRomNo9L, 10 pt, Not Bold
Formatted: Font: NimbusRomNo9L
Formatted: Justified
Formatted: jsrPara
Formatted: Font: NimbusRomNo9L
Formatted: Justified
Formatted: jsrPara
Formatted: Font: NimbusRomNo9L
Formatted: Font: NimbusRomNo9L
Formatted: Font: NimbusRomNo9L
Formatted: Font: NimbusRomNo9L
Formatted: Justified
Formatted: Font: NimbusRomNo9L
Formatted: Font: NimbusRomNo9L
Formatted: Font: NimbusRomNo9L
Formatted: Font: NimbusRomNo9L
Formatted: Font: NimbusRomNo9L
Formatted: Font: NimbusRomNo9L
Formatted: Font: NimbusRomNo9L
Formatted: Font: NimbusRomNo9L
Formatted: Font: NimbusRomNo9L

power consumption from complex input features of large designs. In our research, we aimed to utilize the advantages of non-linear machine learning models to predict power, but our modeling technique took a different analysis at RTL.

Technical Approach

~~In order to~~To predict a design's power consumption, machine learning algorithms were trained with data that includes: (1) features extracted from gate-level netlists, and (2) detailed power usage of the design. The features include type, count and properties of library cells in the given netlists, and are used as training data. The other information used as training data is the report from the power analysis tool. Once the training is complete, the ML model can be used for predicting the power consumption. Figure 1 captures the training flow. The use of ML model for estimating the power consumption of designs is captured in Figure 2. A detailed discussion of the process is presented in this section.

Machine learning algorithms need large sets of data to learn from and identify patterns. Our full design collection amounted to 1,032 behavioral Verilog designs. These designs were from standard benchmarks, such as International Symposium on Circuits and Systems '85 [11] & '89 [12], International Test Synthesis'99 [13], Logic Synthesis and Optimization Benchmarks '89 & '91 [14], and International Workshop on Logic Synthesis'89 [15] & '05 [16]. We used about 80% of these designs for training purpose. Our goal was for the model to find a correlation between the design's features and their power consumption. In this process, Design Compiler [17], the synthesis tool from the Synopsys Suite, was used to synthesize each behavioral Verilog design in the training set. The designs were compiled under some default constraints and optimizations, and their gate-level netlists were created.

4

Formatted: jsrPara, Indent: First line: 0"
Formatted: Font: NimbusRomNo9L, 10 pt
Formatted: jsrH1
Formatted: Font: NimbusRomNo9L, 10 pt, Not Italic
Formatted: Font: NimbusRomNo9L
Formatted: Font: NimbusRomNo9L
Formatted: Justified
Formatted: Font: NimbusRomNo9L
Formatted: Font: NimbusRomNo9L
Formatted: Font: NimbusRomNo9L
Formatted: Font: NimbusRomNo9L
Formatted: Justified, Indent: First line: 0.5"
Formatted: Font: NimbusRomNo9L
Formatted: Font: NimbusRomNo9L
Formatted: Font: NimbusRomNo9L
Formatted: Font: NimbusRomNo9L
Formatted: Font: NimbusRomNo9L
Formatted: Font: NimbusRomNo9L
Formatted: Font: NimbusRomNo9L
Formatted: Font: NimbusRomNo9L
Formatted: Font: NimbusRomNo9L
Formatted: Font: NimbusRomNo9L
Formatted: Font: NimbusRomNo9L
Formatted: Font: NimbusRomNo9L
Formatted: Font: NimbusRomNo9L
Formatted: Justified

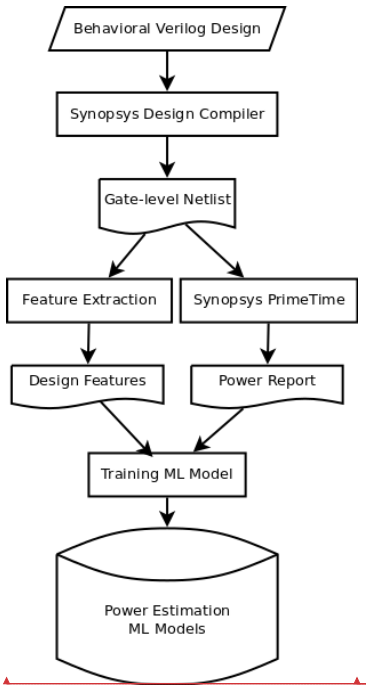

Figure 1 Training the ML Algorithms for Power Estimation

In power analysis mode, PrimeTime [6], the timing and power analysis tool from Synopsys suites is used to compute *internal*, *switching*, *leakage*, and *total power* for each design. Across all designs, a clock period of 10 units, defined by the cell library, was instantiated; otherwise, default timing and power constraints were applied. These designs were compiled into gate-level netlists. Power analysis is performed with a 90 nm, 45 nm, and 15 nm cell library, and corresponding power reports were generated. Each compilation was independent of others. Only full elaborated gate-level netlists and error-free power reports were used in the machine learning models, otherwise the design was excluded from learning data.

Extracting Features (Cell Group Partitioning)

The design features that were extracted and used as training data are primarily based on the type, properties and number of the gates in the gate-level netlist of the design. Two different groupings of the gates were considered. One grouping is entirely based on functionality, and logic level properties such as gate fan-in. As an example, common cell groups were inverters, buffers, and flip-flops. Different types of combinational gates were further divided into subgroups based on their fan-in. In the second grouping, a mapping of the gates to the corresponding library cells is used. The cells were divided based on their average leakage power in the cell library into several groups. *Library Compiler* from Synopsys Suite was used to calculate the cell's average leakage across each state. As leakage power was the only power attribute to be extracted from the cell library, other power metrics were not considered for feature

5

Formatted: Font: NimbusRomNo9L
Formatted: Font: NimbusRomNo9L
Formatted: Justified, Space After: 0 pt
Formatted: Font: NimbusRomNo9L
Formatted: Font: NimbusRomNo9L
Formatted: jsrPara
Formatted: Font: NimbusRomNo9L
Formatted: Font: NimbusRomNo9L
Formatted: Font: NimbusRomNo9L
Formatted: Justified
Formatted: jsrPara, Indent: First line: 0"
Formatted: Font: NimbusRomNo9L, 10 pt
Formatted: jsrPara
Formatted: Font: NimbusRomNo9L
Formatted: Font: NimbusRomNo9L, 10 pt
Formatted: Font: NimbusRomNo9L
Formatted: Justified

extraction. Gates were placed into one of the several groups on this basis. These design features extracted from the gate-level netlist, together with the design's power consumption values (internal, switching and leakage) extracted from the power report, provided the training data to the ML model. These algorithms focus on producing results that can correlate the design features and power values at higher levels of abstraction.

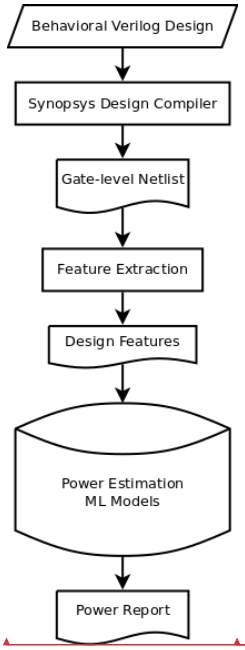

Figure 2. Power Estimation with ML Models

Machine Learning Models

In our experimental set up, four different supervised learning algorithms were tested on each set of training data:

- Multi-Layer Perceptron (MLP) [18]
- Ridge Regression [19]
- Elastic Net [20]
- K-Nearest Neighbors [21]

Scikit-learn [22], an open-source machine learning library in Python, built, trained, and tested the machine learning models. For each algorithm, one model was trained to predict a design's internal, switching, and leakage power, and

Formatted: Font: NimbusRomNo9L, 10 pt
Formatted: jsrPara
Formatted: Font: NimbusRomNo9L
Formatted: Font: NimbusRomNo9L
Formatted: jsrPara, Left
Formatted: Font: NimbusRomNo9L, 10 pt
Formatted: Font: NimbusRomNo9L, 10 pt
Formatted: Font: NimbusRomNo9L, 10 pt
Formatted: Justified, Space After: 0 pt
Formatted: Font: NimbusRomNo9L, Font color: Black
Formatted: jsrPara
Formatted: Font: NimbusRomNo9L
Formatted: Justified
Formatted: Font: NimbusRomNo9L, 10 pt
Formatted: jsrPara
Formatted: Font: NimbusRomNo9L
Formatted: Justified
Formatted: jsrPara
Formatted: Font: NimbusRomNo9L
Formatted: Font: NimbusRomNo9L
Formatted: Justified
Formatted: Font: NimbusRomNo9L
Formatted: Font: NimbusRomNo9L
Formatted: Font: NimbusRomNo9L
Formatted: Font: NimbusRomNo9L
Formatted: Pattern: Clear, Highlight
Formatted: Font: NimbusRomNo9L
Formatted: Normal, Justified, No bullets or numbering
Formatted: Justified
Formatted: Font: NimbusRomNo9L
Formatted: Font: NimbusRomNo9L
Formatted: Indent: First line: 0"

a separate model was trained to solely predict a design's total power. PrimeTime defines a design's total power as the sum of its internal, switching and leakage power. Thus, total power was predicted in a separate model, due to their direct relationship. Additionally, the cross-validation functions were implemented to search for the optimal hyperparameters in each model's training session. In other words, the hyperparameters were reoptimized in each experiment.

Training the Models

Before training the machine learning models, the data underwent preprocessing. First, to remove any outliers: a design was excluded from the data if any of its features or power values were less than or greater than three standard deviations of the entire data set. Next, extremely small and big designs were excluded. In our case, designs with a total cell count of under 30 cells and above 10,000 were excluded. Reasons for this step will be explained in the discussion section. Third, remaining featured columns with no variance were removed. Then, the data was partitioned into two subsets: training data (80%) and testing data (20%) from a random seed. Each model went through 10,000 iterations of fitting to find the optimal seed for the random training and testing split. Because cross validation was used, a separate validation set was not needed. Finally, the features of the training and testing set were standardized. Throughout these iterations, the model with the best symmetric mean absolute percentage error was saved and picked so the model could be tested on another data set, without retraining it. Through the training process, the models applied a mean square error loss function. In order to compare the results, the mean squared error, mean absolute error, and symmetric mean absolute percent error were calculated from the designs in the testing set. These error metrics showed the difference between the model's predicted power values (internal, switching, leakage, and total power) and PrimeTime's calculations of the power values.

Results

Experimental results strongly support that our machine learning methodology can be successfully applied for predicting design's power consumption. Our experiments pointed flaws in our initial set up, and led to ways to improve the accuracy.

Phase I: In initial experiments, one copy of each ML model was used to predict power components (internal, switching and leakage), and another copy of that same model to predict overall power. The accuracy of this set up is captured in tables 1 and 2. A symmetric absolute percent error (SMAPE), which measures the relative percent accuracy of the ML models, is applied to compare the estimated power consumption of the Verilog designs with a standard power consumption calculated by PrimeTime. Table 1 and Table 2 present the errors in predicting internal, switching, leakage, and total power over the different ML models and technology nodes. KNN was the most accurate model at predicting total power in the 90 nm technology, with a 17.91% error. In the 15 nm technology, MLP was the most accurate model at predicting total power with a 32.58% error. The two types of design features, ~~functionality~~functionality and internal power, are averaged in this table, while Table 3 compares the two design features.

Table 1. SMAPE of Predicting Power Consumption on 90 nm Technology.

Prediction Value	Elastic Net	KNN	MLP	Ridge
Internal	63.69%	36.80%	101.38%	51.68%
Leakage	44.26%	17.74%	87.82%	18.88%
Switching	63.81%	30.97%	130.03%	67.39%

~~7~~

Formatted: Font: NimbusRomNo9L
Formatted: jsrPara
Formatted: Font: NimbusRomNo9L, 10 pt
Formatted: Font: NimbusRomNo9L
Formatted: Justified
Formatted: Font: NimbusRomNo9L, 10 pt, Font color: Black
Formatted: jsrPara
Formatted: Font: NimbusRomNo9L
Formatted: Font: NimbusRomNo9L, 8 pt, Not Bold
Formatted: Font: NimbusRomNo9L
Formatted: Justified
Formatted: jsrPara
Formatted: Justified
Formatted: Font: NimbusRomNo9L
Formatted: Font: NimbusRomNo9L
Formatted: Font: NimbusRomNo9L
Formatted: Font: NimbusRomNo9L
Formatted: Font: NimbusRomNo9L
Formatted: Font: NimbusRomNo9L
Formatted: Font: NimbusRomNo9L
Formatted: Font: NimbusRomNo9L, 8 pt
Formatted: jsrPara
Formatted: Font: NimbusRomNo9L, Not Bold, Italic, Font color: Black
Formatted: jsrPara, Don't keep with next
Formatted: Font: NimbusRomNo9L
Formatted: Font: NimbusRomNo9L
Formatted: Font: NimbusRomNo9L
Formatted: Justified, Space After: 0 pt
Formatted: Justified
Formatted: Font: NimbusRomNo9L
Formatted: Justified
Formatted: Font: NimbusRomNo9L
Formatted: Justified
Formatted: Font: NimbusRomNo9L
Formatted: Justified

Total Power	41.08%	17.91%	43.87%	29.68%

Table 2. SMAPE of Predicting Power Consumption on 15 nm Technology

Prediction Value	Elastic Net	KNN	MLP	Ridge
Internal	115.38%	94.60%	53.01%	112.97%
Leakage	100.29%	91.32%	143.70%	100.25%
Switching	130.06%	100.64%	60.35%	129.43%
Total Power	104.84%	86.89%	32.58%	105.45%

Table 3. Error in Predicting Total Power from Partitioning Cells by Functionality and Leakage Power

Partition Type	Elastic Net		KNN		MLP		Ridge	
	15 nm	90 nm	15 nm	90 nm	15 nm	90 nm	15 nm	90 nm
Functionality	103.33%	45.63%	85.61%	17.91%	33.91%	52.09%	99.91%	38.62%
Leakage Power	106.34%	36.53%	88.16%	17.91%	31.26%	35.66%	110.99%	20.74%

We included results for 90 nm and 15 nm technologies. Models that predicted in 90 nm technology preformed the best, while those 15 nm were worse. The results from all nodes can be seen in Figure 3, which plots the estimated total power of designs against PrimeTime's calculation.

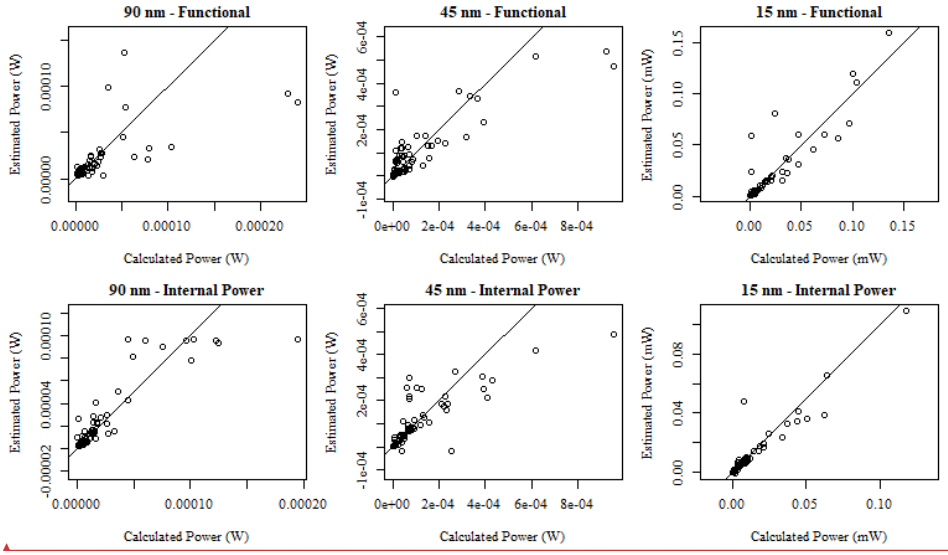

Figure 3. Calculated vs Estimated Total Power of MLP Models for All Technology Nodes and Cell Partitioning Types

Phase II: While analyzing the results presented above, it became clear that treating the three power components (internal, switching and leakage) as the 3 outputs of the same model was not a good decision, as it did not allow the ML tool to establish good correlations between design features and a particular component of the power, while the total power was estimated more accurately in an independent model. Hence, a new set of experiments were performed using three exact copies of our ML Model, each estimating one of the three power components. This significantly improved the results, compared to multi-output components. Table 4 presents this separate experiment, when only one power component (internal, switching, or leakage) was predicted in a single model. In this table we presented the result for MLP model for 15 nm library. The results demonstrate that the accuracy of all power components dramatically increased when they were predicted in individual models. We are optimistic, that our future research can lead to more insight into differences in performance of various ML models in predicting power.

Table 4. Error in Predicting Total Power Consumption in 15 nm Technology, using a Single Output ML Model

Prediction Value	MLP	Net Error between Multiple Output Model
Internal	38.45%	-16.85%
Leakage	48.81%	-83.05%
Switching	39.61%	-15.45%

Discussion

The metrics shown in the results section are from various ML models testing on a set of data they had never seen before. Having a train-test split of the data ensured that the ML models did not 'remember' the correct value from its training process. In the initial experiments for developing the models, we started with a very high mean absolute percent error. Over countless iterations and tweaks to the models, we were able to bring the over 1000% error down to as low as 17.74%.

In the testing stage, total power was the most accurate value predicted compared to leakage, switching and internal powers. We found that running the ML algorithms with only one label (output) leads to significantly more accurate results compared to having three labels (outputs): internal, switching, and leakage powers.

The 90 nm and 45 nm technology libraries used in these experiments were real technology nodes, and the 15 nm library is an ~~open source~~open-source library created solely for research and development purposes. Although the specific 45 nm library we used is also an ~~open source~~open-source library, it aligns with its generation.

As a result of these experiments, we found that the elastic net, k-nearest neighbors, and ridge regression models had the most accurate predictions in the 90 nm library, but they had significantly worse predictions in both the 45 nm and 15 nm libraries. Despite having the highest inaccuracy in the 90 nm library, the MLP had the best results in the 15 nm library, at 31.26%. The 45 nm experimental results were not presented here, as they did not offer any new findings.

When partitioning the models into groups to help the ML models interpret the data easier, we tested two different methodologies. We first partitioned the cells into groups with similar functionalities, then into groups with similar power consumption at the lower (library cell) level, and found that there was no significant statistical difference

~~9~~

Formatted: Justified
Formatted: Font: NimbusRomNo9L
Formatted: Font: NimbusRomNo9L
Formatted: Justified, Space After: 0 pt
Formatted: Font: Not Bold, Italic
Formatted: jsrPara
Formatted: Font: NimbusRomNo9L
Formatted: Font: NimbusRomNo9L
Formatted: Font: NimbusRomNo9L
Formatted: Justified, Space After: 0 pt
Formatted: Justified, Space After: 0 pt, Line spacing: single
Formatted Table
Formatted: Font: NimbusRomNo9L
Formatted: Centered
Formatted: Font: NimbusRomNo9L
Formatted: Centered
Formatted: Font: NimbusRomNo9L
Formatted: Centered
Formatted: Font: NimbusRomNo9L, 10 pt, Not Bold
Formatted: jsrPara
Formatted: Font: NimbusRomNo9L
Formatted: Font: NimbusRomNo9L, Not Bold
Formatted: Font: NimbusRomNo9L, 10 pt, Not Bold
Formatted: Font: NimbusRomNo9L
Formatted: Justified
Formatted: Font: NimbusRomNo9L
Formatted: Font: NimbusRomNo9L

between the two grouping methodologies. However, there was significant difference in the accuracy of the model depending on the number of groups that were considered.

In evaluating the data set, it was noted that most designs are small to medium size with lower power consumption. This trend can be seen in Figure 3 and is a direct contributor to the error metric. These figures also show that the most efficient designs were most accurately predicted, while the least power efficient designs were predicted the worst. This can be attributed to the sporadic nature of switching power consumption. By introducing more data points, the k-nearest neighbors would have less distance between each neighbor, therefore reducing how much it has to approximate.

The symmetric mean absolute percentage error (SMAPE), mean absolute error (MAE), and mean squared error (MSE) of the models were used to accurately compare the different machine learning algorithms. SMAPE was used instead of the mean absolute percentage error (MAPE), to reduce the number of instances where the scoring equation had to divide by zero. The model's score could only return an error if both the true and predicted power values are zero, and because nearly every circuit design consumes at least some power, this situation is a very rare occasion. Using these metrics allowed us to ~~fairly and accurately score each model~~score each model fairly and accurately across each cell library, given that each library had different units of power.

Formatted: Font: NimbusRomNo9L
Formatted: Font: NimbusRomNo9L
Formatted: Font: NimbusRomNo9L
Formatted: Font: NimbusRomNo9L
Formatted: Justified

~~10~~

Conclusion

In this paper, we have demonstrated a ML approach to predicting power consumption in digital systems from their gate-level netlists. We made comparisons from 3 different technology nodes (90 nm, 45 nm, and 15 nm) and 4 different supervised learning models (MLP, ridge regression, elastic net, and k-nearest neighbors). Our MLP models were most successful on the 15 nm cell library, and they significantly improved in accuracy when each power component ("internal", "switching", and "leakage") were predicted with independent models.

References

[1] P. Landman, "Low-power architectural design methodologies," *Ph.D. Thesis, UC Berkeley,* 2019.

[2] J. Carloni, M. Kwon and L. Ziegler, "A learning-based recommender system for autotuning design flows of industrial high-performance processors," in *Proceedings of the 56th Design Automation Conference (DAC)*, 2019.

[3] E. Barboza, N. Shukla, Y. Chen and J. Hu, "Machine learning-based pre-routing timing prediction with reduced pessimism," in *Proceedings of the 56th Design Automation Conference (DAC)*, 2019.

[4] J. Twigg, E. Torkelson and N. Mansouri, "Predicting Formal Verification Resource Needs," *Journal of Student Research (JSR)},* vol. 10, no. 4, November 2021.

[5] G. Singh, J. Gómez-Luna, G. Mariani, G. Oliveira, S. Corda, S. Stuijk, O. Mutlu and H. Corporaal, "NAPEL: Near-memory computing application performance prediction via ensemble learning," in *Proceedings of the 56th Design Automation Conference*.

[6] Synopsys, "PrimeTime," 2019. [Online]. Available: https://www.synopsys.com/implementation-and-signoff/signoff/primetime.html.

[7] "Synopsys," July 2019. [Online]. Available: https://www.synopsys.com.

[8] K. Buyuksahin and F. Najm, "Early power estimation for VLSI circuits," *IEEE Transactions on Computer-Aided Design of Integred Circuits and Systems,* vol. 24, no. 7, pp. 1076-1088, 2005.

[9] J. Yang, L. Ma, K. Zhao, Y. Cai and T. Ngai, "Early stage real-time SoC power estimation using RTL instrumentation," in *Proceedings of the 20th Asia and South Pacific Design Automation Conference (ASPDAC)*, 2015 .

[10] Y. Zhou, R. Haoxing, Y. Zhang, B. Keller, B. Khailany and Z. Zhang, "PRIMAL: Power inference using machine learning," in *In Proceedeings of the 56th Design Automation Conference (DAC)*, 2019.

[11] F. Brglez and H. Fujiwara, "A Neutral Netlist of 10 Combinational Benchmark Circuits and a Target Translator in Fortan," in *Proceedings of the International Symposium of Circuits and Systems*, 1985.

[12] F. Brglez, D. Bryan and K. Kozminski, "Combinational Profiles of Sequential Benchmark Circuits," in *Proceedings of the International Symposium of Circuits and Systems*, 1989.

[13] S. Davidson, "ITC'99 Benchmark Circuits - Preliminary Results," in *Preceedings of International Test Conference*, 1999.

[14] S. Yang, "Logic Synthesis and Optimization Benchmarks User Guide Version 3.0," Microelectronics Center of North Carolina, 1991.

[15] K. McElvain, "IWLS'93 Benchmark Set: Version 4.0," in *International Workshop on Logic & Synthesis*, 1993.

[16] C. Albrecht, "IWLS 2005 Benchmarks," in *International Workshop on Logic Synthesis*, 2005.

~~11~~

Formatted: Font: NimbusRomNo9L
Formatted: jsrH1
Formatted: Font: NimbusRomNo9L, 10 pt
Formatted: Font: NimbusRomNo9L
Formatted: Justified
Formatted: Font: NimbusRomNo9L, 10 pt
Formatted: jsrPara
Formatted: Font: NimbusRomNo9L
Formatted: Font: NimbusRomNo9L, Not Bold
Formatted: Font: NimbusRomNo9L
Formatted: Space After: 0 pt, Line spacing: single
Formatted: Space After: 0 pt, Line spacing: single
Formatted: Space After: 0 pt, Line spacing: single
Formatted: Space After: 0 pt, Line spacing: single
Formatted: Space After: 0 pt, Line spacing: single
Formatted: Space After: 0 pt, Line spacing: single
Formatted: Space After: 0 pt, Line spacing: single
Formatted: Space After: 0 pt, Line spacing: single
Formatted: Space After: 0 pt, Line spacing: single
Formatted: Space After: 0 pt, Line spacing: single
Formatted: Space After: 0 pt, Line spacing: single
Formatted: Space After: 0 pt, Line spacing: single
Formatted: Space After: 0 pt, Line spacing: single
Formatted: Space After: 0 pt, Line spacing: single
Formatted: Space After: 0 pt, Line spacing: single
Formatted: Space After: 0 pt, Line spacing: single

[17] Synopsys, "Design Compiler," July 2019. [Online]. Available: https://www.synopsys.com/implementation-and-signoff/rtl-synthesis-test/dc-ultra.html.

[18] "MLP Regressor," Scikit Learn, [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.neural_network.MLPRegressor.html.

[19] "Ridge," Scikit Learn, [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.Ridge.html.

[20] "Elastic Net," Scikit Learn, [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.ElasticNet.html.

[21] "K Neighbors Regressor," Scikit Learn, [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsRegressor.html.

[22] "SciKit learn: Machine Learning in Python," 2019. [Online]. Available: https://scikit-learn.org/stable/index.html.

Formatted: Space After: 0 pt, Line spacing: single

Formatted: Space After: 0 pt, Line spacing: single

Formatted: Space After: 0 pt, Line spacing: single

Formatted: Space After: 0 pt, Line spacing: single

Formatted: Space After: 0 pt, Line spacing: single

Formatted: Space After: 0 pt, Line spacing: single

Formatted: Space After: 0 pt, Line spacing: single

Formatted: Space After: 0 pt, Line spacing: single

Formatted: Font: NimbusRomNo9L

~~12~~