PRIMAL: Power Inference using Machine Learning

Yuan Zhou‡, Haoxing Ren‡, Yanqing Zhang‡, Ben Keller†, Brucek Khailany†, Zhiru Zhang*

*Cornell University ‡NVIDIA Corporation
{yz882, zhiruz}@cornell.edu {haoxingr, yanqingz, benk, bkhailany}@nvidia.com

Abstract

This paper introduces PRIMAL, a novel learning-based framework that enables fast and accurate power estimation for ASIC designs. PRIMAL trains machine learning (ML) models with design verification testbenches for characterizing the power of reusable circuit building blocks. The trained models can then be used to generate detailed power profiles of the same blocks under different workloads. We evaluate the performance of several established ML models on this task, including ridge regression, gradient tree boosting, multi-layer perceptron, and convolutional neural network (CNN). For average power estimation, ML-based techniques can achieve an average error of less than 1% across a diverse set of realistic benchmarks, outperforming a commercial RTL power estimation tool in both accuracy and speed (15x faster). For cycle-by-cycle power estimation, PRIMAL is on average 50x faster than a commercial gate-level power analysis tool, with an average error less than 5%. In particular, our CNN-based method achieves a 35x speed-up and an error of 5.2% for cycle-by-cycle power estimation of a RISC-V processor core. Furthermore, our case study on a NoC router shows that PRIMAL can achieve a small estimation error of 4.5% using cycle-approximate traces from SystemC simulation.

CCS Concepts

• Computing methodologies → Machine learning; Modeling methodologies; • Hardware → Power estimation and optimization;

Keywords

Power estimation, machine learning

ACM Reference Format:

1 Introduction

Modern VLSI design requires extensive optimization and exploration in a large design space to meet the ever-stringent requirements with respect to performance, area, and power. Existing ASIC CAD tools can provide reasonably accurate area and performance estimates at register transfer level (RTL) or even behavioral level with the aid of high-level synthesis (HLS) tools. However, in order to achieve power closure, designers must obtain detailed power profiles for a diverse range of workloads from different application use cases or even from different levels of design hierarchy. Currently, the common practice is to feed gate-level netlist and simulation results to power analysis tools such as Synopsys PrimeTime PX (PTPX) to generate cycle-level power traces. Figure 1a depicts a typical ASIC power analysis flow, which offers accurate estimates but runs at a very low speed (in the order of 10–100s of cycles per second). Given the high complexity of present-day ASIC designs, it can take hours or days to perform gate-level power analysis for one intellectual property (IP) core under desired workloads.

An alternative is to analyze power above gate level. There exists a rich body of research on power analysis at RTL or a higher abstraction level [3, 4, 6, 7, 14, 20, 21, 23, 24]. These efforts typically make use of measured constants or simple curve fitting techniques such as linear regression to characterize the power of a given circuit, improving the speed of power analysis at the expense of estimation accuracy. For accurate power characterization, many low-level details of the circuit need to be modeled, including standard cell parameters, sizing of the gates, and clock gating status of the registers. Gate-level power analysis uses them to estimate the switching capacitance and activity factor of each circuit node. However, these low-level details are unavailable at (or above) RTL by design. It is also very difficult for simple analytical models or linear regression models to capture the complex nonlinear relationship between the register toggles and the total switching capacitance.

To enable fast and accurate high-level power estimation, we propose PRIMAL, a learning-based power inference framework that enables fast and accurate power characterization of reusable IP cores at RTL or behavioral level. PRIMAL leverages gate-level power analysis to train machine learning (ML) models on a subset of verification testbenches. These trained models can then be used to generate detailed power profiles of the same blocks under different workloads. We evaluate the performance of several established ML models on this task, including ridge regression, gradient tree boosting, multi-layer perceptron, and convolutional neural network (CNN). For average power estimation, ML-based techniques can achieve an average error of less than 1% across a diverse set of realistic benchmarks, outperforming a commercial RTL power estimation tool in both accuracy and speed (15x faster). For cycle-by-cycle power estimation, PRIMAL is on average 50x faster than a commercial gate-level power analysis tool, with an average error less than 5%. In particular, our CNN-based method achieves a 35x speed-up and an error of 5.2% for cycle-by-cycle power estimation of a RISC-V processor core. Furthermore, our case study on a NoC router shows that PRIMAL can achieve a small estimation error of 4.5% using cycle-approximate traces from SystemC simulation.

**Figure 1:** Conventional ASIC power estimation flow vs. PRIMAL — (a) With existing tools, designers must rely on slow gate-level power analysis for accurate power profiles. (b) PRIMAL trains ML-based power models for reusable IPs. Using the trained models, detailed power traces are obtained by running ML model inference on RTL or timed SystemC simulation traces.
used to infer power profiles of the same IP core under different user-specified workloads. Figure 1b illustrates the inference flow of PRIMAL, which only requires inputs from RTL or SystemC simulation to rapidly generate accurate power estimates (>1k cycles per second). By greatly reducing the required number of gate-level simulation cycles, PRIMAL allows designers to perform power-directed design space exploration in a much more productive manner. The major technical contributions of this work are fivefold:

- We present PRIMAL, a novel ML-based methodology for rapid power estimation with RTL or timed SystemC simulation traces. The trained ML models can provide accurate, cycle-by-cycle power inference for user workloads even when they differ significantly from those used for training.
- We investigate several established ML models for power estimation, and report trade-offs between accuracy, training effort, and inference speed. Our study suggests that nonlinear models, especially convolutional neural nets (CNNs), can effectively learn power-related design characteristics for large circuits.
- We explore feature engineering techniques to construct image representations from register traces. The constructed features are used by CNNs for training and inference.
- We demonstrate that PRIMAL is at least 50x faster on average than PTPX for cycle-accurate power estimation with a small error. Notably, our CNN-based approach is 35x faster than PTPX with a 5.2% error for estimating power of a RISC-V core. PRIMAL also achieves a 15x speedup over a commercial RTL power analysis tool for average power estimation.
- Using a NoC router design as a case study, we demonstrate that PRIMAL can be extended to enable accurate power estimation for timed SystemC design.

The remainder of this paper is organized as follows: Section 2 surveys related work, and Section 3 presents the overall design methodology and intended use cases of PRIMAL. Section 4 introduces our feature construction methods. Experimental results are reported in Section 5. Section 6 gives concluding remarks.

2 Related Work

Power estimation is an extensively studied research topic. Existing works focus on power estimation from behavioral-level and RTL. Behavioral-level power estimation provides optimization guidance early in the design flow. In an earlier work, Chen et al. [7] combine profiling and simple analytical models to estimate FPGA power consumption. Later efforts use the HLS tool to perform scheduling and back-annotation, and rely on RTL power analysis [3], gate-level power analysis [21], or a per-control-step ML power model [14] for power estimation.

Compared to behavioral-level analysis, more implementation details are available at RTL. Earlier works in RTL power estimation use simple regression models, such as linear regression and regression trees, to characterize small circuit blocks [4, 6, 20]. The regression models are trained with gate-level power analysis results. Average power and cycle-by-cycle power of the whole design can be obtained by summing up the outputs from multiple models. PrEsto [23] uses linear models to characterize larger modules, where heavy feature engineering and feature selection are applied to reduce the complexity of power models. A more recent work by Yang et al. [24] uses a single linear model to characterize the whole design. A feature selection technique based on singular value decomposition (SVD) is applied to reduce model complexity so that the regression model can be efficiently mapped onto an FPGA. Both PrEsto and [24] can provide cycle-by-cycle power estimates.
Figure 3: Simple circuit example with waveform of register outputs and register connection graph.

Figure 4: Basic feature encoding methods — (a) 1D switching encoding. (b) Default 2D encoding for edge 1 in Figure 3b.

4 Feature Construction

This section describes the feature construction procedure using the circuit in Figure 3a as an example. Figure 3b shows the register waveform, where each "edge" in the figure corresponds to a clock rising edge. We use register switching activities in the simulation traces as input features, because register switching activities are representative of the circuit’s state transitions. In addition, there is a one-to-one correspondence between registers in RTL and gate-level netlist. Because we use cycle-accurate power traces from gate-level simulation as ground truth, the ML models are essentially learning the complex relationship between the switching power for all gate-level cells and register switching activities. For clarity we focus on RTL power estimation in this section, but our feature construction methods can also be naturally applied or extended to SystemC power estimation.

Feature Encoding for Cycle-by-Cycle Power Estimation

For cycle-by-cycle power estimation, we use RTL register and I/O signal switching activities as input features without any manual feature selection. Switching activities of both internal registers and I/O signals are required to capture complete circuit state transitions. These features can be easily collected from RTL simulation. Because we are targeting cycle-by-cycle power estimation, each cycle in the simulation trace is constructed as an independent sample.

A good feature encoding should differentiate between switching and non-switching events. A concise encoding, which we refer to as switching encoding, is to represent each register switching event as a 1, and non-switching event as a 0. For an RTL module with \(n\) registers, each cycle in the RTL simulation trace is represented as a \(1 \times n\) vector. Figure 4a shows the corresponding encoding for the waveform in Figure 3b. Each vector in Figure 4a represents one cycle (one clock rising edge to be precise) in the waveform. We use this one-dimensional (1D) switching encoding for all but the CNN models. The same feature encoding is used in [24].

In order to leverage well-studied two-dimensional (2D) CNN models, we create a three-channel 2D image representation for every cycle in the register trace. For an RTL module with \(n\) registers, we use a \(\sqrt{n} \times \sqrt{n}\) image to encode one cycle in the RTL simulation trace. We use one-hot encoding in the channel dimension to represent the switching activities of each register: non-switching is represented as \([1, 0, 0]\), switching from zero to one is represented as \([0, 1, 0]\), and switching from one to zero is \([0, 0, 1]\). We refer to this encoding as default 2D encoding. Figure 4b shows how we encode edge 1 of the waveform in Figure 3b. The total number of pixels in the image is greater than \(n\), we add padding pixels to the image, shown as \(d’s\) in Figure 4b. These padding pixels do not represent any register in the module, and they have zero values in all three channels in our implementation. Every other pixel corresponds to one register in the module. For this default 2D encoding, the registers are mapped by their sequence in the training traces. For example, since in Figure 3b the order of registers is \(A, B, C, D, E\), in each channel the top-left pixel in Figure 4b corresponds to \(A\), the top-right pixel is mapped to \(C\), and the center pixel refers to \(E\). We observe that this default mapping from registers to pixels is not completely random. The tool flow we are using would actually cluster most of the registers within a submodule together. As a result, in our experiments, this default 2D encoding actually preserves a considerable amount of circuit structural information.

Mapping Registers and Signals to Pixels

In the default 2D encoding described above, the mapping between registers and pixel locations are determined by the way the registers are arranged in the trace file. The amount of structural information that is preserved is dependent on the tool flow. As a result, this mapping method cannot guarantee meaningful local structures in the constructed images. Registers that are mapped to adjacent pixels may not be correlated or physically connected. CNNs are most effective when there are spatial relationships in their 2D inputs. Therefore, the register-to-pixel mapping should reflect the connectivity or physical placement of the registers. Since the gate-level netlist of the design is available during the characterization phase, it is possible to use the outputs of logic synthesis tools to map RTL registers to netlist nodes. Because we only use register and I/O switching activities, we ignore all combinational components and only extract register connection graphs when processing the gate-level netlist. The graph for the example circuit in Figure 3a is shown in Figure 3c. Each node
We use Table 1: Benchmark information — We evaluate PRIMAL with a diverse set of benchmark designs. For NoC router and RISC-V core, the test sets are realistic workloads which are potentially different from the corresponding training set.

<table>
<thead>
<tr>
<th>Design</th>
<th>Description</th>
<th>Register + I/O signal count</th>
<th>Gate count</th>
<th>PTPX throughput (cycles/s)</th>
<th>Training set (cycles)</th>
<th>Test set (cycles)</th>
</tr>
</thead>
<tbody>
<tr>
<td>qadd_pipe</td>
<td>32-bit fixed point adder</td>
<td>160</td>
<td>838</td>
<td>1250</td>
<td>Random stimulus (480k)</td>
<td>Random stimulus (120k)</td>
</tr>
<tr>
<td>qmult_pipe[1, 2, 3]</td>
<td>32-bit fixed point multiplier with 1, 2, or 3 pipeline stages</td>
<td>[384, 405, 438]</td>
<td>[1721, 1718, 1749]</td>
<td>[144.9, 135.1, 156.3]</td>
<td>Random stimulus (480k)</td>
<td>Random stimulus (120k)</td>
</tr>
<tr>
<td>float_adder</td>
<td>32-bit floating point adder</td>
<td>381</td>
<td>1239</td>
<td>714.3</td>
<td>Random stimulus (480k)</td>
<td>Random stimulus (120k)</td>
</tr>
<tr>
<td>float_mult</td>
<td>32-bit floating point multiplier</td>
<td>372</td>
<td>2274</td>
<td>454.5</td>
<td>Random stimulus (480k)</td>
<td>Random stimulus (120k)</td>
</tr>
<tr>
<td>NoCRouter</td>
<td>Network-on-chip router for a CNN accelerator</td>
<td>5651</td>
<td>15076</td>
<td>44.7</td>
<td>Unit-level testbenches (910k)</td>
<td>Convolution tests (244k)</td>
</tr>
<tr>
<td>RISC-V Core</td>
<td>RISC-V Rocket Core (SmallCore)</td>
<td>24531</td>
<td>80206</td>
<td>45</td>
<td>RISC-V ISA tests (2.2M)</td>
<td>RISC-V benchmarks (1.7M)</td>
</tr>
</tbody>
</table>

* Use random 50% training data per training epoch.

in the graph corresponds to one register in the design, and two nodes are connected if their corresponding registers are connected by some combinational data path.

We propose two graph-based methods for generating register-to-pixel mappings, which introduce local structures into the images according to the structural similarities between nodes. Notice that the proposed graph-based mapping methods only change the register mapping in the width and height dimensions of the image: we still use the channel-wise one-hot encoding for every register. Each register’s contribution to each pixel is proportional to the overlapping area of the register’s occupied region and the pixel. In other words, with the graph-based encoding methods, the pixel values are non-negative real numbers rather than binary numbers.

The first method is based on graph partitioning, in which the graph is recursively divided into two partitions of similar sizes, and the partitions are mapped to corresponding regions in the image (see Figure 5a). The area allocated for each partition is computed according to the number of nodes in the partition. The second method is based on node embedding. Node embedding techniques map each node in the graph to a point in a vector space, where similarity between nodes is expressed as a distance in the vector space. Our flow for embedding-based register mapping is shown in Figure 5b. We use node2vec [10] for node embedding, then apply PCA [12] and t-SNE [16] to project the vector representations to 2D space. The resulting 2D vector representations are scaled according to the image size and indicate the mapping locations of the registers.

5 Experiments
We have implemented our proposed framework in Python 3.6, leveraging networkx [11],metis [13], and a node2vec package [10]. MLP and CNN models are implemented using Keras [2]. Other ML models are realized in scikit-learn [19] and XGBoost [8]. We conduct our experiments on a server with an Intel Xeon E5-2630 v4 CPU and a 128GB RAM. We run neural network training and inference on an NVIDIA 1080Ti GPU. The SystemC models of our designs are synthesized with Mentor Catapult HLS. We use Synopsys Design Compiler for RTL and logic synthesis, targeting a 16nm FinFET standard cell library. The RTL register traces and gate-level power traces are obtained from Synopsys VCS and PTPX, respectively. Gate-level power analysis is performed on another server with an Intel Xeon CPU and 64GB RAM using a maximum of 30 threads.

5.1 Benchmarks
Table 1 lists the benchmarks used to evaluate PRIMAL. Our benchmarks include a number of fixed- and floating-point arithmetic units from [18]. We also test our approach against two complex designs — a NoC router used in a CNN accelerator and a RISC-V processor core. The NoC router block is written in SystemC and synthesized to RTL by an HLS tool. The RISC-V core is an RV64IMAC implementation of the open-source Rocket Chip Generator [5] similar to the SmallCore instance. We use different portions of random stimulus traces as training and test sets for the arithmetic units. For the NoC router and the RISC-V core, we select functional verification testbenches for training and use realistic workloads for test. For the NoC router, we test on actual traces of mesh network traffic from a CNN accelerator SoC. In the RISC-V experiment, dhrystone, median, multiply, qsort, qsort, and vavd form the set of test workloads.

5.2 RTL Power Estimation Results
Figure 6 summarizes the results for RTL power estimation. Here we use RTL register traces as the raw input and apply the feature construction techniques described in Section 4. Two percent of the training data is used as a validation set for hyper-parameter tuning of the ML models. They are also used for early stopping when training the deep neural networks.

All models except CNNs use the 1D switching encoding, while CNNs use the 2D image encoding methods introduced in Section 4. We also experimented with 1D one-hot encoding, where we encode the switching activity of each register using three binary numbers as described in Section 4. Since the results with such encoding is similar with 1D switching encoding, we omit the results due to space limitations. For ridge regression and gradient tree boosting, we apply PCA to reduce the size of input features to 256, except for qadd_pipe which has only 160 features with 1D feature encoding. We use three-layer MLP models for the arithmetic unit and four-layer MLP models for the NoC router and the RISC-V core. We use an open-source implementation [1] of ShuffleNet V2 [15] for CNN-based power estimation because of its parameter-efficiency architecture and fast inference speed. The v0.5 configuration in [15] is used for the arithmetic units, while the v1.5 configuration is used for the NoC router and RISC-V core. The CNN models are trained from scratch. CNN-default, CNN-partition, and CNN-embedding in Figure 6 refer to the default 2D encoding, graph-partition-based register mapping, and node-embedding-based register mapping methods introduced in Section 4, respectively.

Cycle-by-Cycle Power Estimation Results — We use normalized root-mean-squared-error (NRMSE) as our evaluation metric. Suppose the ground-truth power trace is represented as a n-dimensional vector y, and the estimated power trace is a vector ŷ.
with default 2D encoding achieves an impressive 5.2% error on the
As shown in Figure 6a, all ML models can achieve an average
ORS competitive accuracy with much less training e
XGBoost o
To
gure out the latent space of toggling registers and show it in
samples. Essentially, with the 2D encoding methods, we are trying
cult for clustered onto a small number of pixels, making it more di
graph-based encoding methods, information of multiple registers is
accuracy on some benchmarks. One possible reason is that with the
default encoding. In fact, the advanced encodings result in a lower
fault 2D encoding because of the rich structural information in the
register mapping methods do not provide much bene
Complex non-linear functions. We observe that the graph-based
clustered onto a smaller number of pixels, making it more difficult for
CNN models to distinguish between high-power and low-power
samples. Essentially, with the 2D encoding methods, we are trying to
figure out the latent space of toggling registers and show it in
of the same shape. Then
\[
\text{NRMSE} = \frac{1}{\bar{y}} \sqrt{\frac{1}{n} \sum_{i=1}^{n} (y_i - \hat{y}_i)^2}
\]
As shown in Figure 6a, all ML models can achieve an average estimation error of less than 5% across our benchmarks. The training
time for each ML model is summarized in Table 2. For small designs,
XGBoost offers competitive accuracy with much less training effort.
CNN models show significant advantage over other ML models for larger designs like the RISC-V core. Notably, our CNN model
with default 2D encoding achieves an impressive 5.2% error on the
test set, while MLP, XGBoost and Linear model achieves around
8%, 11% and 13% error, respectively. Noticeably, the estimation error for the RISC-V core is considerably higher than other designs.
Compared with other benchmarks, the RISC-V core contains ~5x more gates but has only five pipeline stages. Each pipeline
stage is significantly more complex and harder to model. In addition, it is harder to create a comprehensive training set for the models
to approximate a larger design.

Figure 7 compares the estimation of CNN-default and PCA+Linear with the ground truth power trace. The CNN estimation fits the ground truth curve more closely. These results demonstrate the superior capability of deep neural networks in approximating complex non-linear functions. We observe that the graph-based
register mapping methods do not provide much benefit over default 2D encoding because of the rich structural information in the
default encoding. In fact, the advanced encodings result in a lower accuracy on some benchmarks. One possible reason is that with the
graph-based encoding methods, information of multiple registers is clustered onto a small number of pixels, making it more difficult for
the CNN models to distinguish between high-power and low-power
samples. Essentially, with the 2D encoding methods, we are trying to
figure out the latent space of toggling registers and show it in

Figure 6: Performance of different machine learning models on test sets — The ML models used by PRIMAL achieve high accuracy for both cycle-by-cycle and average power estimation, while offering significant speedup against both Synopsys PTPX and the commercial RTL power analysis tool (Comm). PRIMAL is also significantly more accurate than Comm in average power estimation.

RISC-V — Showing 300 cycles from the dhrystone benchmark.
the two-dimensional space. This problem itself is an interesting
research direction, and we leave it for future work.

Average Power Estimation Results — The average power consumption for a workload can be easily obtained from a cycle-
accurate power trace. We compare the ML-based techniques with a commercial RTL power analysis tool (Comm). According to Figure 6b,
all of the ML techniques achieve less than 1% average error, while the commercial tool has an average error of 20%. NoCRouter has higher error for average power estimation, because the training set and the test set have very different average power. Interestingly, while the CNN models achieve similar or higher accuracy compared with other ML models for cycle-by-cycle power estimation, their accuracy for average power estimation is slightly worse because the CNN models tend to consistently overestimate or underestimate power by a very small margin. This behavior may be caused by a mismatch in average power between the training and test sets: the CNN models learn the average power of the training set better, causing a small yet consistent shift in their estimations for the test set. The ML models require a significant amount of time to be trained for complex designs as shown in Table 2. Therefore, the commercial RTL power analysis tool is still favorable for power estimation of non-reusable modules.

Speedup — Figure 6c presents the speedup of the commercial RTL power analysis tool and the PRIMAL techniques against Synopsys.
error, which is satisfactory in most cases. Nevertheless, the error of SystemC power estimation remains higher than that of RTL power estimation because of the trace inaccuracy and the information loss in the feature construction process.

6 Conclusions
We have presented PRIMAL, a learning-based framework that enables fast and accurate power estimation for ASIC designs. Using state-of-the-art ML models, PRIMAL can be applied to complex hardware such as a RISC-V core, and the trained power models can generalize to workloads that are dissimilar to the training benchmarks. The ML-based techniques achieve less than 5% and 1% average error for cycle-by-cycle and average power estimation, respectively. Compared with Synopsys PTPX, PRIMAL provides at least 50X speedup across our selection of benchmarks. We also demonstrate that PRIMAL can be readily extended to SystemC power estimation through a case study on NoC router.

Acknowledgements
This research was supported in part by NSF Award #1512937 and the Intel ISRA Program. We thank the anonymous reviewers for providing suggestions and helpful feedback to our work.

References