Unleashing Flexibility of ML-based Power Estimators Through Efficient Development Strategies

Yao Lu*, Qijun Zhang*, Zhiyao Xie† Hong Kong University of Science and Technology {yludf, qzhangcs}@connect.ust.hk, eezhiyao@ust.hk

(*equal contribution, †corresponding author)

ABSTRACT

Power is a primary design objective in modern VLSI design. Efficient and accurate power evaluation tools are in high demand to provide prompt power feedback for early design optimization. However, it is time-consuming to simulate long, fine-grained (e.g., per-cycle) power traces in complex designs with commercial power simulators. In recent years, machine learning (ML)-based power models have emerged as a potential solution to make fast predictions on per-cycle power based on signal toggling activities. Despite their immense potential, these models are currently underutilized in realistic development scenarios, largely due to the challenges associated with their development and updates. To overcome the barriers, this work optimizes the often-neglected power model development process by an in-depth examination of power data's impact on model accuracy. We propose efficient strategies to minimize the overhead involved in model development. Furthermore, we enhance model flexibility by enabling the transfer of existing models to updated design RTLs with negligible additional costs.

CCS CONCEPTS

• Hardware \rightarrow Power estimation and optimization.

KEYWORDS

Power modeling and estimation, machine learning

1 INTRODUCTION

In recent years, various data-driven power modeling techniques [9, 14, 15, 19, 20] have been proposed to make fast fine-grained (e.g., per-cycle) power predictions. They develop efficient machine learning (ML) models based on data patterns, which are extracted from RTL signal-toggling activities in *.fsdb* or *.vcd* file formats. Many power models [14, 19, 20] aim at efficient power simulation at design time¹, while some [9, 14, 15] are further implemented onchip for runtime power management. By adopting lightweight ML models, these data-driven power models generally achieve orders-of-magnitude speedup over commercial tools [3, 13]with $\leq 10\%$ average error. They can help estimate fine-grained power traces for large and realistic workloads like SPEC in a very short time.

Despite obvious speed advantages demonstrated in prior works, ML-based power models are still seldom adopted in realistic IC development scenarios as an alternative power modeling option. We attribute this largely to the inflexibility and unclearness in the

ISLPED '24, August 5-7, 2024, Newport Beach, CA, USA

© 2024 Copyright held by the owner/author(s). Publication rights licensed to ACM. ACM ISBN 979-8-4007-0688-2/24/08

https://doi.org/10.1145/3665314.3670823

Power Models*	Overhead for New Design	Accuracy	Inference Speed	Per-Cycle Power	
Design-Specific Models [9, 14, 15, 20]	High	High	Fast	Yes	
General Power Models [8, 12, 16]	Zero	Low	Fast	No	
Commercial Tools: PTPX [13], PowerPro[3]	Zero	Very High	Slow	Yes	
AgileDevelop AgileTransfer	Very Low	High	Fast	Yes	

*Here we focus on summarizing power simulators/models at RTL or downstream design stages. The architectural power models [7, 10, 17, 18] are not included in the table.

Table 1: Summary of Power Estimators. AgileDevelop and AgileTransfer achieves low development overhead, high ac curacy, fast inference speed, and high temporal resolution.

power model development process. To ensure high accuracy, most ML-based power models [9, 14, 15, 20] are design-specific, which means a unique ML model has to be trained from scratch for each specific design. Such design-specific models are much more accurate than recent general ML solutions [8, 12, 16], but require developing new models very frequently. In addition, existing research efforts [9, 14, 15, 19, 20] mostly only focus on designing better power models, without inspecting the power model developers and users. Frequently asked questions include: "How many labeled data shall I collect? What workload should be used to collect training data? There is an update in the design RTL, shall I retrain the power model?"

Challenges: We have observed three unsolved challenges that widely exist in the development process of ML-based power models:

- In a typical development process, designers will keep revising and optimizing their designs. Such design updates will easily make the design-specific power model inapplicable.
- (2) The development of data-driven power models requires accurate fine-grained power simulation results as training labels. The generation of detailed ground-truth power labels by simulating complete workloads is highly time-consuming.
- (3) There is no clear guidance on the data requirement during the model development process. As a result, developers tend to collect much more power labels than needed.

In summary, almost all existing power modeling works simply assume a fixed design, largely neglecting the fact that circuit designs will keep evolving during the design optimization process, requiring more flexible power models. To make things worse, the limited understanding of training data's impact further makes developers collect as many power labels as possible, and the label generation process is time-consuming. For example, accurate per-cycle power

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

¹Strictly speaking, GRANNITE [19] is an ML-based average toggle rate propagation method, instead of a power simulator itself.

simulation [13] on a single academic RISC-V CPU core using one testbench with around 50 thousand cycles takes much more than 24 hours. The label generation cost is expected to be higher for more complex industrial designs.

Goal and Solution: In this work, we aim to build power models efficiently by in-depth inspection of the seldom-explored model development process. We propose efficient development strategies to build flexible power models for two scenarios: 1) When developing a new power model from scratch, we propose a general strategy named *AgileDevelop* to greatly reduce the development cost. Specifically, when generating per-cycle power labels as training data, we only need less than 1% of representative labeled data instead of simulating the whole workload. 2) Based on an existing design-specific power model, when there is an update in design RTL, we propose to transfer the power model to the updated new design with negligible overhead. This method is named *AgileTransfer*.

Rationales: Our proposed solution is based on a key observation: there is heavy duplication of signal toggling activities between different clock cycles. For a given workload with *N* cycles, there exist many clock cycles where the signal toggling activities of all *d* RTL signals are similar. As a result, it is not necessary to simulate power labels for all *N* cycles. In this work, we demonstrate that a power model can be developed with negligible accuracy loss by only using less than 1% representative labeled cycles, which means less than 1% data generation time.

Contributions: Contributions of this work are summarized below.

- This is a pioneering work that investigates the seldom-explored development process of power models and proposes efficient power model development strategies².
- For any target design, we propose a new data sampling strategy *AgileDevelop*, which develops ML-based power models based on a tiny amount of labeled data. We demonstrate that the absolute error percentage < 3% based on only 50 cycles of labeled data, less than 0.1% of the whole workload.
- Based on an existing power model, we propose a new strategy *AgileTransfer*, which efficiently transfers it to a similar new design. We demonstrate that the absolute error percentage ≈ 2% based on only 1 cycle of labeled data.
- We advance the understanding of the power data's impact on the power model. We investigated the effect of sample coverage in both feature and label spaces, and also reported the impact of available workloads on the model accuracy.

2 POWER MODEL OVERVIEW

In this section, we give an overview of the development process of ML-based power models, as shown in Figure 1. The input of power model \mathcal{M} is the toggling of all of the RTL signals, and the output is the predicted per-cycle power. To build the model, the dataset is N-cycle toggling activity of all RTL signals, denoted as $D = \{c_i\}_{i=1}^N$, where each sample c_i represents the toggling in one cycle. The sample $c_i \in \mathbb{R}^d$ is a binary vector, whose length d represents the total number of signals in the design. Following prior work [14], we train linear power models using minimax concave penalty (MCP). In this paper, we target the power model applied at design time, where a linear power model is sufficiently fast.

As Figure 1 shows, the power model features are per-cycle toggling from *RTL simulation*, while labels are accurate per-cycle power from *power simulation*. Since the RTL simulation and model training are fast, the power model development speed is dominated by



Figure 1: The power model development framework. the power simulation time, which is proportional to the number of simulated cycles (i.e. labeled data samples). Therefore, to efficiently build the model, we propose *AgileDevelop* and *AgileTransfer*, aiming at minimizing required labeled data while maximizing model accuracy. We formulate *AgileDevelop* and *AgileTransfer* below:

AgileDevelop: Given an unlabeled dataset $D = \{c_i\}_{i=1}^N$, where $c_i \in \mathcal{R}^d$, with N cycles and d RTL signals, we design a strategy AgileDevelop to select some data Samples from dataset D to be labeled and then used for model training. Our goal is to design a strategy AgileDevelop to minimize the number of sampled data [Samples] while maximizing the accuracy:

max Accuracy, min |Samples|

Accuracy, Samples \leftarrow AgileDevelop(D) (1) $D = \{c_i\}_{i=1}^N, c_i \in \mathcal{R}^d$

AgileTransfer: Given a source design s with labeled dataset $D_s = \{c_i^s\}_{i=1}^{N_s}$, where $c_i^s \in \mathcal{R}^{d_s}$, with N_s cycles and d_s RTL signals, and target design t with unlabeled dataset $D_t = \{c_i^t\}_{i=1}^{N_t}$, where $c_i^t \in \mathcal{R}^{d_t}$, with N_t cycles and d_t RTL signals, we design a strategy AgileTransfer to transfer the model built on labeled source design dataset D_s to target design, and then build a model for target design with some data Samples selected from dataset D_t to be labeled. Our goal is to design a strategy AgileTransfer to minimize the number of sampled data |Samples| while maximizing the accuracy:

(max Accuracy, min |Samples|

$$Accuracy, Samples \leftarrow AgileTransfer(D_s, D_t)$$
(2)
$$D_s = \{c_i^s\}_{i=1}^{N_s}, c_i^s \in \mathcal{R}^{d_s}, D_t = \{c_i^t\}_{i=1}^{N_t}, c_i^t \in \mathcal{R}^{d_t}$$

3 MODEL DEVELOPMENT FROM SCRATCH

In this section, we introduce the *AgileDevelop* method, which outlines how to select cycles within a single design, as shown in the green part in Figure 1. Our approach is primarily based on the analysis of signals and cycles, and it employs selection strategies to reduce labeling overhead. Each cycle is viewed as an independent sample in the dataset. The *AgileDevelop* is included in two parts, Basic Sampling Method in Session 3.1 and Coverage-based Sampling in Session 3.2.

3.1 Basic Sampling Method

Given that samples within the training dataset exhibit significant similarities, without labels, we employ unsupervised techniques to identify the most representative samples. The basic method first performs dimension reduction for d RTL signals as model features, followed by clustering-based sampling among all N-cycle samples.

Due to the large number of RTL signals *d*, the resulting feature space becomes highly dimensional. The curse of dimension poses challenges for evaluating distance or similarity between data samples. Therefore, unsupervised dimension reduction becomes essential, as shown in the line 5 in the Algorithm 1. This unsupervised

²It is open-sourced in https://github.com/hkust-zhiyao/AgileDevelop

Algorithm 1 AgileDevelop: Develop Power Model from Scratch

Input: The unlabeled dataset $D = \{c_i\}_{i=1}^N$, where $c_i \in \mathbb{R}^d$, with N cycles and d RTL signals. The target number of cycles to be sampled K. **Hyperparameters**: the dimension after reduction d', the number of initial samples N_I .

Output: The power model \mathcal{M} .

/* Line 1-4 are algorithms of using clustering for selection*/

- 1: **function** INITSELECT (D', N_I)
- Clustering D' into N_I clusters 2:
- Select the sample closest to the center of each cluster as Samples 3:
- return Samples 4:

/* Line 5 performs dimension reduction */

- 5: Reduce dimension of *D* to $D' = \{c'_i\}_{i=1}^N$, where $c'_i \in \mathcal{R}^{d'}$ /* Line 6 generates the initial selection, and selected samples will be labeled, denoting the label of *i* as $l_i \in \mathcal{R}^*$
- 6: $Samples = InitSelect(D', N_I)$

/* Line 7-11 are algorithm of coverage-based active learning */ 7: for *iter* = N_I to K do

- Build model \mathcal{M} using Samples 8:
- Make Prediction $\{p_i\}$, where $i \in D' Samples$ using \mathcal{M} 9: /* (1) Line 10 calculates the distance between each unlabeled data and each labeled data, and then selects the unlabeled data with maximum distance to its closest labeled data. (2) We define the distance between any two samples using the multiplication of their features distance and label distance. */ $\underset{i \in D'-Samples}{\operatorname{arg\,max}} \min_{k \in Samples} Dist(c'_i, c'_k) * |p_i - l_k|$ $i_{best} = \arg \max$ 10:

11: Add *i*best into Samples

/* line 12 trains the model using selected samples */

12: Build model *M* using *Samples*

method utilizes the entire training set D for efficient dimension reduction, in the absence of labeled data.

Then we propose a clustering-based sample selection, shown as the InitSelect function in the line 6 in Algorithm 1. We cluster the training dataset into N_I clusters and choose the sample closest to the cluster center as the representative sample for each cluster. This method ensures that we obtain a diverse and representative subset of samples while minimizing redundancy.

3.2 **Coverage-based Sampling**

To further improve the effect of AgileDevelop, we propose an innovative coverage-based sampling method, considering both feature and label spaces. The underlying idea is to sample a labeled dataset that achieves comprehensive coverage of both signal feature space and the power label space. With good coverage of the two spaces, our sampled data can better represent the full dataset and yield similar model accuracy after training.

We formulate this sampling process as a K-Center problem in both feature and label space, similar to the set cover problem [11]. Here we give a general formulation. Given N points $X = \{x_i\}_{i=1}^N$ in a space and a positive integer *K*, we aim to select *K* points *Samples* = $\{x_k\}_{k=1}^K$ as centers. The objective is to minimize R, the maximum distance from each remaining point x_i to its nearest center x_k $(k \in Samples)$. The objective can be formulated as $\min_{Samples} R$,

where
$$R = \max_{i \in X-Samples} r_i$$
, $r_i = \min_{k \in Samples} Dist(x_i, x_k)$ (3)

 $Dist(x_i, x_k)$ is distance between two points x_i and x_k .

In our problem, to address both feature space and label space simultaneously, we define the distance between two data samples as the product of the distance in feature and label space:

$$Dist(x_i, x_k) = Dist(c_i, c_k) * |l_i - l_k|$$
(4)



Figure 2: The greedy solution to solve the K-Center problem. For each iteration, the point that "stuck" the largest circle is selected. The ultimate distance is the product of distances in both feature and label spaces.

The K-Center problem can be solved using various methods, and we propose to adopt an efficient greedy algorithm, as illustrated in Figure 2. The detail is shown as line 7-11 in Algorithm 1. Each iteration selects one sample. First, we calculate the distance between each unlabeled data $i \in D' - Samples$ and labeled data $k \in Samples$. Then we select the unlabeled data *i*best with maximum distance to its closest labeled data:

$$i_{best} = \underset{i \in D'-Samples}{\arg \max} \min_{k \in Samples} Dist(c'_i, c'_k) * |l_i - l_k|$$
(5)

As illustrated in Figure 2, the point that "stuck" the largest circle will be selected in each iteration.

However, it should be noted that for the unlabeled data, labels are not available. In such cases, we substitute their labels with predictions $\{p_i\}$ generated by a temporary model \mathcal{M} trained with the currently selected samples, as illustrated in line 8-9, thus the Equation 5 should be transformed into:

$$i_{best} = \arg\max_{i \in D'-Samples} \min_{k \in Samples} Dist(c'_i, c'_k) * |p_i - l_k|$$
(6)

Because we use the prediction p_i to ensure the effectiveness of the algorithm, a set of initial warm-up samples is selected before the iteration process. This initial selection is carried out using the method described in Section 3.1, as indicated in line 6 of Algorithm 1.

In addition, due to the large number of initial samples N, directly applying coverage-based sampling can be time-consuming. In practice, we employ a preliminary clustering step to select an initial diverse pool of samples, named candidate pool. The coverage-based selection is performed on the candidate pool instead of all N cycles. This approximation greatly enhances the efficiency.

TRANSFERRING MODEL TO NEW DESIGNS 4

Most accurate ML-based power models are design-specific, which means each power model is tailored to a specific design. In each design iteration, when there is an update in the target design RTL, there arises a recurrent need to reconstruct the power model, leading to significant model development overhead. However, when updating a design, most design components usually remain unchanged, and only a small part is updated. Based on this key observation, this section aims to reduce the model updating overhead.

To reduce the cost of model updating, we propose a novel model transfer methodology. It leverages the inherent similarities among different versions of the design. After gathering sufficient data from a known design, we construct a power model on this design named source model. Remarkably, for a target design, only a minimal amount of labeled data is needed to train the target model, based on the knowledge transferred from the source model.

To capture the similarity between the source design and the target design, we utilize common signals between these two designs. The specific algorithm is shown in the Algorithm 2. The unlabeled **Input**: The unlabeled target design dataset $D_t = \{c_i^t\}_{i=1}^{N_t}$, where $c_i^t \in \mathcal{R}^{d_t}$, with N_t cycles and d_t RTL signals. The labeled source design dataset $D_s = \{c_i^s\}_{i=1}^{N_s}$, where $c_i^s \in \mathcal{R}^{d_s}$, with N_s cycles and d_s RTL signals, the target number of cycles to be sampled K

Output: The power model \mathcal{M} of target design.

/* 1 line 1-3 build a transferable model for source design */

- 1: Get the *C* common signals of target and source design
- 2: Select the common signals from D_t and D_s , denoted as $D_t^c = \{c_i^{tC}\}_{i=1}^{N_t}$ and $D_s^c = \{c_i^{sC}\}_{i=1}^{N_s}$, where $c_i^{tC}, c_i^{sC} \in \mathcal{R}^C$
- 3: Build model \mathcal{M}_{src} using D_s^c
 - /* 2 line 4 generates pseudo label for target design */
- 4: Generate initial pseudo label $L_I = \{I_i^I\}_{i=1}^{N_t}$ by applying \mathcal{M}_{src} on D_t^c /* ③ line 5-11 calibrate pseudo label for target design */
- 5: Get *Samples* by *AgileDevelop*, with label denoted as $L_g = \{l_i^g\}_{i=1}^K$
- 6: Group D_t to the nearest sample in the *K* labeled data, as $\{G_i\}_{i=1}^K$
- 7: for Each group G_i do
- 8: Compute the discrepancy $k_i = l_i^g / l_i^I$
- 9: **for** Each sample j in G_i **do**
- 10: Calibrate the pseudo label $l_j^P = k_i * l_j^I$

/* line 12 trains the model using calibrated pseudo label */ 11: Build model \mathcal{M} using D_t and $L_P = \{l_i^P\}_{i=1}^{N_t}$

target design dataset is $D_t = \{c_i^t\}_{i=1}^{N_t}$, where $c_i^t \in \mathcal{R}^{d_t}$, with N_t cycles and d_t RTL signals. The labeled known source design dataset is $D_s = \{c_i^s\}_{i=1}^{N_s}$, where $c_i^s \in \mathcal{R}^{d_s}$, with N_s cycles and d_s RTL signals. There are three steps in *AgileTransfer*:

In step ①, *C* common signals are identified based on their signal name and hierarchical location, and two datasets with only *C* common signals are generated: $D_t^c = \{c_i^{tC}\}_{i=1}^{N_t}$ and $D_s^c = \{c_i^{sC}\}_{i=1}^{N_s}$, where both $c_i^{tC}, c_i^{sC} \in \mathcal{R}^C$, as shown in line 2. Utilizing labeled dataset D_s^c from the source design, we construct a high-quality model, labeled as \mathcal{M}_{src} , as shown in line 3. The idea is, considering the substantial correlation among signals in a design, as studied in pruning-based power models [14], even a subset of signals can be sufficient for accurate power model construction.

In step O, the source model \mathcal{M}_{src} is directly applied to predict target design D_t^c , generating power predictions for each cycle, used as initial pseudo labels $L_I = \{l_i^I\}_{i=1}^{N_t}$, as shown in line 4.

In step (), we propose to utilize the aforementioned AgileDevelop method to select specific samples for the target design. Power labels of selected samples are obtained from accurate simulations, denoted as $L_g = \{l_i^g\}_{i=1}^K$, as shown in line 5. Based on these sampled labels, we adjust all pseudo labels L_I through calibration. Specifically, for each unlabeled data in D_t , we search for the nearest labeled data *i*. All of the data in D_t , we search for the nearest labeled data *i*. All of the data in D_t whose nearest labeled data is *i* are grouped as G_i , so grouping is denoted as $\{G_i\}_{i=1}^K$, as shown in line 6. For each G_i , we query for the pseudo label l_i^I and ground truth power l_i^g for the only labeled data *i*, then the discrepancy can be computed as $k_i = l_i^g / l_i^I$, as shown in line 8. Then for each data *j* in this group $\{G_i\}$, the pseudo label is calibrated by multiplying it by discrepancy k_i , as shown in line 10, the calibrated pseudo label is denoted as l_j^P . Finally, all samples with calibrated pseudo labels $L_P = \{l_i^P\}_{i=1}^{N_t}$ are

used to train the power model for target design.

5 EXPERIMENTAL RESULTS

5.1 Experiment Setup

We generated a dataset by collecting RTL code and conducting RTL simulations using Chipyard v1.8.1 [6]. We implement three different RISC-V BOOM CPU configurations, namely C_A , C_B , and C_C , as



Figure 3: Per-cycle power accuracy *MAPE* and *R* vs. number of sampled cycles. Applying the methods Random, K-Means, and *AgileDevelop* to C_A and C_B with training data sizes of 50-800, respectively. Add an additional training data size of 1 to *AgileTransfer*($C_B \rightarrow C_A$ in (a)(b) and $C_A \rightarrow C_B$ in (c)(d)).

shown in Table 2. Several common parameters are FetchWidth=4, DTLBEntry=8, DCacheMSHR=2, and ICacheFetchBytes=2. We selected eight different workloads from the RISCV-tests suite [5] to enhance the diversity of the data. These workloads cover a variety of computing tasks, including dhrystone, median, multiply, qsort, rsort, towers, spmv, and vvadd. All evaluation results are obtained by using a four-fold cross-validation method. For each fold, we use six workloads for training and the other two workloads for testing. The total number of cycles for these workloads is approximately 130k, and the baseline model uses all of the cycles.

RTL simulation on workloads is based on Synopsys VCS[®] [4]. The logic synthesis and ground-truth power simulation were executed at a clock frequency of 1GHz using Synopsys Design Compiler[®] [2] and PrimePower [13], respectively. In our experiments, we utilized the TSMC 40nm standard cell library [1], along with the corresponding Memory Compiler. By leveraging these industry-standard tools, we ensured accurate and reliable simulations, enabling us to evaluate the power characteristics of the designs.

In Section 3.1, we discuss the dimensionality reduction technique to address the curse of high dimensionality in the dataset. In our experiments, we employ principal component analysis (PCA) for dimensionality reduction. The clustering is performed using the K-means algorithm.

To comprehensively evaluate our methods, we conduct experiments on different configurations and different numbers of samples to demonstrate the advantage of our method. For the *AgileDevelop*, we conduct our experiment on configuration C_A and C_B respectively, under 50, 100, 200, 400, and 800 samples. For the *AgileTransfer*, we utilize three designs C_A , C_B , and C_C , constructing three pairs of transferring scenarios, i.e. $C_A \leftrightarrows C_B$, $C_B \boxdot C_C$, $C_A \leftrightarrows C_C$, and conduct them under 1, 50, 100, 200, 400, 800 samples.

We evaluate the accuracy with mean absolute percentage error (*MAPE*) and correlation coefficient (*R*) between label Y_i and prediction \hat{Y}_i .

$$MAPE = \frac{1}{n} \sum_{i=1}^{n} \left| \frac{Y_i - \hat{Y}_i}{Y_i} \right| \times 100\%, R = \frac{\sum_{i=1}^{n} (\hat{Y}_i - \hat{Y})(Y_i - \bar{Y})}{\sqrt{\sum_{i=1}^{n} (\hat{Y}_i - \bar{Y})^2 \sum_{i=1}^{n} (Y_i - \bar{Y})^2}}$$

For strategies involving randomness, such as random sampling, we performed multiple tests and calculated the average results to mitigate the impact of randomness. Employing this cross-validation

Configuration Parameter Decode	Width FetchBufferEntry	RobEntry	IntPhyRegister	FpPhyRegister	LDQ/ STQEntry	BranchCount	IntIssueWidth	DCache/ ICacheWay
C _A 1	1 5	16	36	36	4	6	1	2
C _B 2	2 16	64	80	64	16	12	2	4
C _C 1	1 8	32	53	48	8	8	1	4

Table 2: The configurations we used in the experiment.



Figure 4: Comparing method AgileDevelop and AgileTransfer to Label (exact power value) and Baseline on all workloads of C_A . Baseline utilizes a training dataset of 103.3k cycles, while AgileDevelop and AgileTransfer only sampled 50 cycles as training dataset (0.048% of the size used by Baseline)

technique and averaging the results over multiple samples, which provides a robust and reliable measure of its accuracy ensures a fair evaluation of the performance of our power model.

5.2 **Power Prediction Result**

For C_A and C_B , the accuracy of baseline models is R = 0.97, MAPE = 0.59% and R = 0.97, MAPE = 0.98%, respectively. These models trained with complete datasets serve as our baseline for comparison in subsequent analysis.

We start by comparing our methods, AgileDevelop and Agile-Transfer, along with K-means and Random selections. On C_A , as depicted in Figure 3(a)(b), that the *AgileDevelop* achieves R = 0.8and MAPE = 2.26% with 50 sampled cycles, while K-means and Random sampling do not work at all with the same condition. As the number of samples increases to 400, R of AgileDevelop is only 0.01 lower than the Baseline, and MAPE is 1.97%, while at this point, R of K-means and Random are 0.85 and 0.89, respectively. What's more, AgileTransfer achieves amazing results. It reaches R = 0.9and MAPE = 1.87% with only one sampled cycle, and when sampling 50 cycles, R rises to 0.95 and MAPE drops to 1.12%. It suggests that in comparison to the Baseline, we require only 0.048% of the training data to achieve nearly equivalent results. In Figure 3(c)(d), the experiment performed on C_B maintains the same trend as C_A . Particularly noteworthy is the performance of our AgileDevelop and AgileTransfer methods, which achieve comparable results to the Random and K-means methods with just 1-50 cycles of training data. This performance level is equivalent to what the other methods achieve with 400-800 cycles of training data. Both our methods achieve equally impressive results on C_A and C_B , demonstrating its universality and potential for extension to other designs.

AgileDevelop and *AgileTransfer* reduce the model development overhead significantly. Specifically, for the baseline, the power simulation to get labels for all workloads with about 100k cycles takes about 3 days. Compared to the power simulation time, the time consumption of RTL simulation and model building is negligible.

With *AgileDevelop*, to build an effective power model, we just need to conduct power simulation for 100 cycles, so we can reduce



Figure 5: A more obvious comparison of per-cycle power among methods *AgileDevelop*, *AgileTransfer* and *Label*. Zoom in the timing window of C_A with index = [70000:70050].



Figure 6: A comparison of the average power over a time window with 8 cycles among methods *AgileDevelop*, *AgileTransfer*, and *Label*. Zoom in the timing window of C_A with index = [70000:70050].

the power simulation time to roughly only 10 minutes. Please notice that this measurement is performed on a single-core academic CPU design. In more complex industrial designs with larger and more diverse workloads, the unprecedented efficiency contributed by *AgileDevelop* and *AgileTransfer* will be more obvious and necessary. For example, prior works [14] reported a much slower power label generation speed for industrial designs.

Figure 4 visualizes predicted results (*Baseline, AgileDevelop*, and *AgileTransfer*) and actual power values (*Label*). Compared with *Baseline*, the overall power trace shape of *AgileDevelop* and *AgileTransfer* are very similar, with a reduction in *R* of only 0.17 and

Metrics		R ↑			MAPE(%) ↓					
Num Scenario	1	50	100	400	800	1	50	100	400	800
$C_A \rightarrow C_B$	0.83	0.91	0.91	0.94	0.95	2.53	2.07	2.00	1.61	1.43
$C_B \rightarrow C_A$	0.90	0.95	0.95	0.96	0.96	1.87	1.12	1.10	0.98	0.90
$C_A \rightarrow C_C$	0.85	0.91	0.91	0.94	0.96	2.01	1.69	1.65	1.41	1.17
$C_C \rightarrow C_A$	0.86	0.93	0.94	0.96	0.96	1.79	1.28	1.18	1.00	0.92
$C_B \rightarrow C_C$	0.93	0.93	0.94	0.95	0.96	1.70	1.54	1.50	1.30	1.14
$C_C \rightarrow C_B$	0.87	0.94	0.94	0.94	0.95	2.18	1.58	1.54	1.50	1.36
Average	0.87	0.93	0.93	0.95	0.95	2.01	1.55	1.50	1.30	1.15

Table 3: Metrics for *AgileTransfer*. We construct six transferring scenarios, i.e. $C_A \leftrightarrows C_B, C_A \leftrightarrows C_C$, and $C_B \leftrightarrows C_C$, and conduct them under 1, 50, 100, 200, 400, 800 samples.

0.02, and a decrease in *MAPE* of 1.67% and 0.53%, respectively. *Baseline* utilizes a training dataset of 103.3k cycles, while *AgileDevelop* and *AgileTransfer* only sampled 50 cycles (0.048% of 103.3k), which demonstrates the effectiveness of our methods.

For better visibility, we present a rigorous comparison by zooming in per-cycle waveforms in Figure 5. In this experiment, each cycle is only 1 *ns*, and the waveform of *AgileDevelop* roughly matches the Label, while *AgileTransfer* exhibits an even better alignment. In Figure 6, we slightly relax the experimental conditions and estimate the average power over a time window with 8 cycles. It can be observed that both our methods exhibit a more pronounced consistency with the waveform of the *Label*.

Table 3 lists all six transfer scenarios, including $C_A \rightarrow C_B$ and $C_B \rightarrow C_A$, which are represented by green lines in Figure 3. The other four scenarios also conform to the same trend. When only utilizing one sample, *R* and *MAPE* can reach 0.87 and 2.01% on average. And as the number of samples gradually increases, the accuracy quickly approaches the baseline.

6 **DISCUSSION**

6.1 Coverage in Both Spaces Matters

To fully evaluate the advantage of *AgileDevelop*, we compare it with two variants when just sampling 50 data on configuration C_A and C_B for ablation study: (1) BasicSampling: the basic sampling method described in Session 3.1. This comparison evaluates the advantage of coverage-based sampling over basic clustering-based sampling. (2) Coverage—: weak version of Coverage-based sampling, only using the feature coverage $Dist(c'_i, c'_k)$, without label coverage $|p_i - l_k|$.

Table 4 clearly shows that $\hat{AgileDevelop}$ outperforms BasicSampling in terms of both R and MAPE for C_A and C_B . This result provides strong support for the effectiveness of optimizing coverage rather than relying solely on clustering. Additionally, AgileDevelop performs better than Coverage—, indicating that considering only feature coverage is inadequate and the significance of label coverage should not be ignored.

6.2 Dataset Size and Diversity Matter

We also evaluate how the amount and diversity of initial unlabeled samples (i.e., size of sampling space N) affect the model accuracy. We try to only use one workload to generate the initial dataset D. Results are shown in Table 5. When using one workload as the

Metrics	R ↑		MAP	E(%)↓
Configuration	C_A	CB	C_A	C_B
BasicSampling	0.27	0.67	5.77	3.54
Coverage	0.72	0.72	2.60	3.29
AgileDevelop	0.80	0.80	2.26	2.75

Table 4: An ablation study comparing *AgileDevelop* with two extra variants (BasicSampling and Coverage—) when sampling 50 data on configuration C_A and C_B .

Metrics	R↑				MAPE(%)↓			
Num of sampled cycles	50	100	800	All	50	100	800	All
Average of each workload	0.58	0.73	0.90	0.92	3.43	2.43	1.34	0.91
All workloads	0.80	0.84	0.96	0.97	2.26	1.97	0.92	0.59

Table 5: Sampling from one workload vs. from all workloads. dataset, the accuracy is lower compared to using the full dataset, regardless of whether sampling is performed or not. It indicates that besides the number of sampled labels, the search space size and diversity also affect model accuracy. Therefore, we suggest preparing sufficient workloads before starting the sampling methods.

7 CONCLUSION

In this work, we propose *AgileDevelop* and *AgileTransfer*, efficient ML-based power model development strategies. *AgileDevelop* minimizes the overhead involved in model development from scratch by coverage-based sampling, encompassing both signal space and power space. *AgileTransfer* enables the transfer of existing models to updated design RTLs with negligible additional costs by generating pseudo labels through source design datasets and calibrating them with sampled ground truth power. The strategies significantly reduce the label generation bottleneck from 3 days to 10 minutes while maintaining a high accuracy. Such lightweight development and transfer strategies greatly lower the barrier to adopting ML-based power models by design teams in realistic application scenarios.

ACKNOWLEDGEMENT

This work is supported by the National Natural Science Foundation of China (NSFC) 62304192.

REFERENCES

- 2008. TSMC 40nm LP process technology. https://www.tsmc.com/english/dedicated Foundry/technology/logic/l_40nm.
- [2] 2021. Design Compiler® RTL Synthesis. https://www.synopsys.com/ implementation-and-signoff/rtl-synthesis-test/design-compiler-nxt.html.
- [3] 2021. PowerPro® RTL Low-Power. https://www.mentor.com/hls-lp/powerprortl-low-power/.
- [4] 2021. VCS® functional verification solution. https://www.synopsys.com/ verification/simulation/vcs.html.
- [5] 2022. RISC-V Tests. https://github.com/riscv-software-src/riscv-tests
- [6] Alon Amid et al. 2020. Chipyard: Integrated design, simulation, and implementation framework for custom socs. *IEEE Micro* (2020).
- [7] David Brooks, Vivek Tiwari, and Margaret Martonosi. 2000. Wattch: A framework for architectural-level power analysis and optimizations. ACM SIGARCH Computer Architecture News (2000).
- [8] Wenji Fang, Yao Lu, Shang Liu, Qijun Zhang, Ceyu Xu, Lisa Wu Wills, Hongce Zhang, and Zhiyao Xie. 2023. MasterRTL: A Pre-Synthesis PPA Estimation Framework for Any RTL Design. In *ICCAD*.
- [9] Donggyu Kim et al. 2019. Simmani: Runtime power modeling for arbitrary RTL with automatic signal selection. In *MICRO*.
- [10] Sheng Li, Jung Ho Ahn, Richard D Strong, Jay B Brockman, Dean M Tullsen, and Norman P Jouppi. 2009. McPAT: an integrated power, area, and timing modeling framework for multicore and manycore architectures. In *MICRO*.
- [11] Ozan Sener and Silvio Savarese. 2018. Active Learning for Convolutional Neural Networks: A Core-Set Approach. In ICLR.
- [12] Prianka Sengupta, Aakash Tyagi, Yiran Chen, and Jiang Hu. 2022. How Good Is Your Verilog RTL Code? A Quick Answer from Machine Learning. In *ICCAD*.
- [13] Synopsys. 2023. PrimePower: RTL to Signoff Power Analysis. https://www. synopsys.com/implementation-and-signoff/signoff/primepower.html
- [14] Zhiyao Xie et al. 2021. APOLLO: An automated power modeling framework for runtime power introspection in high-volume commercial microprocessors. In MICRO.
- [15] Zhiyao Xie et al. 2022. DEEP: Developing Extremely Efficient Runtime On-Chip Power Meters. In ICCAD.
- [16] Ceyu Xu, Chris Kjellqvist, and Lisa Wu Wills. 2022. SNS's not a synthesizer: a deep-learning-based synthesis predictor. In ISCA.
- [17] Jianwang Zhai, Chen Bai, Binwu Zhu, Yici Cai, Qiang Zhou, and Bei Yu. 2021. McPAT-Calib: A microarchitecture power modeling framework for modern CPUs. In 2021 IEEE/ACM International Conference On Computer Aided Design (ICCAD).
- [18] Qijun Zhang, Shiyu Li, Guanglei Zhou, Jingyu Pan, Chen-Chia Chang, Yiran Chen, and Zhiyao Xie. 2023. PANDA: Architecture-Level Power Evaluation by Unifying Analytical and Machine Learning Solutions. In *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*.
- [19] Yanqing Zhang, Haoxing Ren, and Brucek Khailany. 2020. GRANNITE: Graph neural network inference for transferable power estimation. In DAC.
- [20] Yuan Zhou, Haoxing Ren, Yanqing Zhang, Ben Keller, Brucek Khailany, and Zhiru Zhang. 2019. PRIMAL: Power inference using machine learning. In DAC.