APOLLO: An Automated Power Modeling Framework for Runtime Power Introspection in High-Volume Commercial Microprocessors

Zhiyao Xie
Duke University
zhiyao.xie@duke.edu

Xiaoqing Xu
Arm Ltd.
xiaoqingxu.pku@gmail.com

Matt Walker
Arm Ltd.
Matt.Walker@arm.com

Joshua Knebel
Arm Ltd.
joshua.knebel@arm.com

Kumaraguru Palaniswamy
Arm Ltd.
KumaraGuru.Palaniswamy@arm.com

Nicolas Hebert
Arm Ltd.
nicolas.hebert@arm.com

Jiang Hu
Texas A&M University
jianghu@tamu.edu

Huanrui Yang
Duke University
huanrui.yang@duke.edu

Yiran Chen
Duke University
yiran.chen@duke.edu

Shidhartha Das
Arm Ltd.
Shidhartha.Das@arm.com

ABSTRACT

Accurate power modeling is crucial for energy-efficient CPU design and runtime management. An ideal power modeling framework needs to be accurate yet fast, achieve high temporal resolution (ideally cycle-accurate) yet with low runtime computational overheads, and easily extensible to diverse designs through automation. Simultaneously satisfying such conflicting objectives is challenging and largely unattained despite significant prior research.

In this paper, we propose APOLLO, an automated per-cycle power modeling framework that serves as the basis for both a design-time power estimator and a low-overhead runtime on-chip power meter (OPM). APOLLO uses the minimax concave penalty (MCP)-based feature selection algorithm to automatically select less than 0.05% of RTL signals as power proxies. The power estimation achieves $R^2 > 0.95$ on Arm Neoverse N1 [3] and $R^2 > 0.94$ on Arm Cortex-A77 [2] microprocessors, respectively. When integrated with an emulator-assisted flow, APOLLO finishes per-cycle power estimation on millions-of-cycles benchmark in minutes for million-gate industrial CPU designs. Furthermore, the power model is synthesized and integrated into the microprocessor implementation as a runtime OPM. APOLLO's accuracy further improves when coarse-grained temporal resolution is preferred. To our best knowledge, this is the first runtime OPM that simultaneously achieves per-cycle temporal resolution and < 1% area/power overhead without compromising accuracy, which is validated on high-performance, out-of-order industrial CPU designs.

CCS CONCEPTS

• **Hardware → Power estimation and optimization**; *On-chip resource management*.

KEYWORDS

Power modeling and estimation, on-chip power meter, machine learning, voltage droop, commercial microprocessors

ACM Reference Format:
Zhiyao Xie, Xiaoqing Xu, Matt Walker, Joshua Knebel, Kumaraguru Palaniswamy, Nicolas Hebert, Jiang Hu, Huanrui Yang, Yiran Chen, and Shidhartha Das. 2021. APOLLO: An Automated Power Modeling Framework for Runtime Power Introspection in High-Volume Commercial Microprocessors. In *MICRO'21: 54th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO '21), October 18–22, 2021, Virtual Event, Greece.* ACM, New York, NY, USA, 14 pages. https://doi.org/10.1145/3466752.3480064

1 INTRODUCTION

Stringent energy-efficiency demands drive design decisions across the entire compute-spectrum, ranging from embedded applications, mobile computing to data-centers. As such, accurate power estimation is crucial for making prudent engineering trade-offs not only during CPU microarchitecture design [29, 35, 40, 79] but also for runtime power management. The requirements on power estimation differ according to the target application. For instance, dynamic voltage and frequency scaling (DVFS) [33, 81] is orchestrated by the system firmware and/or the operating system (OS), and hence requires *coarse-grained* temporal resolution in power-tracing, where each sample represents power for epochs that can be microseconds in duration.

MICRO '21, October 18–22, 2021, Virtual Event, Greece

ACM ISBN 978-1-4503-8557-2/21/10…$15.00
https://doi.org/10.1145/3466752.3480064

In contrast, recent techniques for fast power management [25, 39] and voltage boosting [32] require fine-grained temporal resolution - for instance, a complete voltage boosting operation in [32] occurs in tens of nanoseconds. Similarly, voltage-noise effects such as Ldi/dt noise develops in <10 cycles in modern high-performance CPUs. Therefore, quantifying the impact of fast voltage-noise and the efficacy of mitigation features such as adaptive-clocking[1] require *fine-grained* temporal resolution in power-tracing [27, 59, 69], where a sample exists for every CPU cycle (per-cycle temporal resolution).

Design-Time Power-Modeling: For fine-grained power-tracing, CPU design teams typically rely on industry-standard power analysis tools such as [8] to replay simulation vectors at the RTL or gate-level with back-annotated parasitics. Power is computed from the switching statistics of individual signal nets and the capacitive load that they drive. This approach is very accurate and serves as the signoff standard, but it comes with a very high computational cost. For instance, generating a power trace of 20 cycles for a power-virus benchmark on the Arm® Neoverse™ N1 microprocessor [21, 57] requires approximately an hour on a high-performance computing cluster machine. Clearly, such an approach does not scale for analysis on long-running workloads and/or simulating the simultaneous execution of multiple CPU cores.

An alternative approach relies upon FPGA-based netlist emulation [7] to address the speed impact of power estimation. In this approach, a simulation trace is generated from FPGA, then the extracted switching statistics are processed using power analysis EDA software [8] to obtain power traces. However, per-cycle power tracing is still onerous using this approach due to the significant storage constraints on modern computer servers. Our own benchmarking studies demonstrate storage requirements in excess of 200GB for a 17-million cycle simulation, leading to infeasible execution time using power analysis tools. Thus, this approach is typically restricted to coarse-grained temporal resolution where power tracing is averaged over millions of CPU cycles.

Runtime Power Estimation: Previous works have demonstrated runtime regression models using hardware performance monitoring event-counters to guide OS-orchestrated DVFS [16, 36, 68]. These models average counter-values that accumulate specific micro-architectural events, such as L2 cache misses and the number of retired instructions, across thousands or millions of CPU cycles. However, these events typically exhibit poor correlations to per-cycle micro-architectural activity. Furthermore, the process of averaging over long CPU cycles renders these approaches significantly inaccurate when fine-grained power tracing is required.

Recently, RTL-based runtime power monitoring with on-chip power meter (OPM) [23, 51, 53, 80, 81] has been proposed to improve temporal resolution at the expense of dedicated hardware circuit. However, existing techniques struggle to simultaneously achieve high resolution and low hardware area overhead. For example, the work in [51] restricts area overhead to 1.5-4%, but its highest temporal resolution is 2500 clock cycles. A recent work [80] improves resolution to 100 cycles, but with significant area overhead

[1] Adaptive-clocking [18, 49] and issue throttling are typically deployed to address timing emergencies due to Ldi/dt transients [69]. Voltage droop due to these transients develop in < 10 cycles in CPU designs. Hence, assessing the performance impact of these micro-architectural features requires fine-grained tracing capabilities.

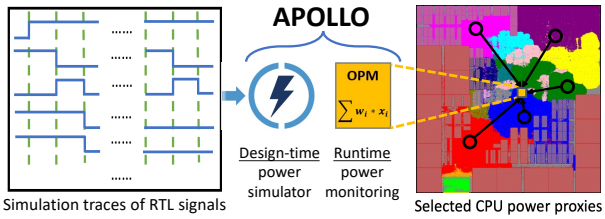


Figure 1: APOLLO provides a design-time power simulator and a runtime on-chip power meter (OPM) based on a consistent model, as an example, for Neoverse™ N1.

(4-10%). Thus, there are undesirable trade-offs between accuracy, speed, temporal-resolution, and on-chip hardware overhead that render the prior-art unsuitable for fine-grained power estimation.

We propose APOLLO, a unified RTL-level power modeling framework addressing both the design-time and runtime challenges within a consistent model structure, as shown in Figure 1. The centerpiece of APOLLO is a new power proxy selection technique based on minimax concave penalty (MCP) regression. It enables per-cycle power tracing for benchmarks executing over millions of CPU cycles. For runtime monitoring, it provides per-cycle accurate power estimation with 0.2% area overhead. To the best of our knowledge, APOLLO is the first power monitoring technique with cycle-accuracy and sub-1% area overhead. Moreover, the proxy selection process in APOLLO is fully automated and thereby extensible to new designs. Compared to PRIMAL [79], a recent machine learning approach, APOLLO reaches similar accuracy but is orders of magnitude faster. APOLLO also significantly outperforms Simmani [40], another state-of-the-art work, on both accuracy and computation speed. Moreover, APOLLO achieves both fine-grained temporal resolution and lower hardware overhead than [53], a recent OPM technique.

Our main contributions in APOLLO are as follows:

- **Fine-grained temporal resolution with low-overhead OPM:** APOLLO provides the first runtime OPM that simultaneously achieves per-cycle resolution and <1% area/power overhead without compromising accuracy.
- **Automation:** The overall framework automatically generates training data, develops the model, and constructs the OPM for an arbitrary novel CPU core with minimum designer interference. This reduces the engineering cost and design time on power estimation.
- **Validation on industrial designs:** APOLLO is verified on industry-standard CPU cores with millions of gates, namely, Arm® Neoverse™ N1 [1, 4, 57] microprocessor and Cortex®-A77 [2] microprocessor. APOLLO achieves an $R^2 > 0.94$ by only monitoring < 0.05% of the RTL signals on both microprocessors.
- **Design-time model with low computational overhead:** The linear APOLLO model finishes per-cycle power estimation on millions of benchmark cycles within minutes when integrated in emulator-assisted power analysis. It enables architects to examine the whole real-world workloads with extremely low cost.

The key novelty of APOLLO over prior-art is the ability to accurately estimate CPU current-demand at a per-cycle temporal

Methods (Hardware Overhead in Area %)	Demonstrated Application	Model Type	Temporal Resolution	PC / Proxy Selection	Cost or Overhead
[20, 35, 43, 48, 61]	Design-time software model	Analytical	>1K cycles	N/A	Low
[78]		Proxies	>1K cycles	Automatic or no selection	High
[17, 64]					Medium
[79]			Per-cycle		High
[19, 42, 44, 72, 76]					Medium
[22] **(300% overhead)**	Design-time FPGA emulation	Proxies	Per-cycle	Automatic	High
[75] **(16% overhead)**					Medium
[40]			~100s cycles		
[66]			Per-cycle	Hybrid manual/auto	
[10, 11, 16, 24, 26, 33, 34, 36, 52, 58, 62, 63, 65, 68]	Runtime monitor	Event Counters	>1K cycles	Manual	Low
[38]			~100s cycles		
[23] **(2-20%)**, [51] **(1.5-4%)**, [53] **(7%)**		Proxies	>1K cycles	Automatic	Medium
[80] **(4-10%)**, [81] **(7%)**			~100s cycles		
APOLLO **(0.2% overhead)**	Design-time model Runtime monitor	Proxies	Per-cycle	Automatic	Low

Table 1: Comparison among various power modeling approaches. The percentage numbers are area overheads.

granularity with <1% power/area overhead, proven in industrial CPU designs. The framework is systematic and automated (including training data generation) in a manner that is micro-architecture agnostic, applicable to a wide spectrum of compute-units and not just CPUs. To the best of our knowledge, APOLLO is the only unified framework for power-modeling achieving combined objectives of low-overhead, high temporal resolution, and accuracy. As we describe in Section 8, these unique features open up new applications in design and runtime power-management beyond the capability of the prior art.

2 RELATED WORK

Table 1 summarizes representative power estimation approaches, which are categorized into design-time power models and runtime on-chip power meters.

2.1 Design-Time Power Models

Many design-time approaches [20, 35, 43, 48, 61] construct analytical models for micro-architectural power estimation by collecting statistics from performance simulators [14, 15]. Wattch [20] is an architectural dynamic power simulation tool using a linear model, and McPAT [48] integrates power, area, and timing in a modeling framework. Each functional unit is characterized and attributed a power value when activated. Multiple active units are then added together to compute the overall power [27]. However, this approach cannot handle internal variations in power consumption due to data- and control-dependent variations in workload. Therefore, these models are preferably used as an average over thousands or millions of CPU clock cycles. Additionally, inaccuracies have been observed [45, 61, 73] for McPAT on new designs.

Design-time models on selected RTL power-proxies are employed to perform power simulations. Early works [17, 72, 76] construct macro-models to abstract power estimations for small circuit modules with thousands of gates. In recent years, machine learning (ML) techniques are exploited. Lee, et al. [44] adopt gradient boosting and Kumar, et al. [42] apply a decision tree model to every component of a simple microprocessor. PRIMAL [79] predicts per-cycle power by processing transitions of all registers with the convolutional neural network (CNN). GRANNITE [78] makes use of graph neural network [41] to estimate the average power of each workload. Although the ML approach achieves significant speedup compared with accurate commercial tools [8], it can be prohibitively expensive (computationally) for per-cycle simulation on industry-standard CPU designs. Evidently, these techniques are intended for simulation-level power-tracing and are too expensive for runtime on-chip monitoring.

FPGA emulation [22, 40, 66, 75] is a popular approach to accelerating power simulations for large designs. We use the term "emulation" in a broad sense to include techniques that use of FPGA at design-time. In reality, there are various ways to do so, which may be named differently in other literature. Perhaps the first power emulation work is [22], which has 300% hardware overhead. Another work [75] employs singular value decomposition (SVD), which can be computationally expensive. Both [22] and [75] are demonstrated only at block-level designs. A microprocessor-level application of FPGA emulation is Simmani [40], whose temporal resolution is 128 clock cycles. PrEsto [66] achieves cycle-accuracy, but its hardware cost is quite significant, e.g., it consumes more than 50% of LUTs on Xilinx Virtex-5 LX330 to simulate ARM Cortex-A8 processor design. Moreover, its proxy selection process is not completely automated.

2.2 Runtime On-Chip Power Meters (OPMs)

Analog power sensors [12, 13] can provide accurate power estimation at runtime. However, they require ADCs that consume a large area overhead. A popular runtime approach is to estimate power dissipation according to performance counters [10, 11, 16, 24, 26, 33, 34, 36, 58, 62, 63, 65, 68]. Since these counters already exist in industrial-grade microprocessor designs, they can be treated as free and the associated area overhead is minimum. However, counter-based methods typically rely on architects' knowledge of a specific design to define representative hardware events. This limits existing methods to well-studied microprocessors and hinders automatic migration to new designs. For example, [10, 16, 34]

Symbol	Description of the symbol
N; M	Number of cycles; Number of RTL signals in design
Q	Number of selected power proxies, $Q \ll M$
S_M; S_Q	All RTL signals $\lvert S_M\rvert = M$; Power proxies $\lvert S_Q\rvert = Q$
$x[i]$	Toggling of proxies at each cycle i, $x[i] \in \{0,1\}^Q$
$\mathbf{x}[i]$	Toggling of RTL signals at each cycle i, $\mathbf{x}[i] \in \{0,1\}^M$
$y[i]$	Label at each cycle i, $y[i] \in \mathbb{R}$
$p[i]$	Predicted power at each cycle i, $p[i] \in \mathbb{R}$
w	Weights of the per-cycle APOLLO power model
ω	Weights of the multi-cycle APOLLO$^\tau$ power model
$\mathcal{L}$; $\mathcal{P}$	Prediction error in loss function; Penalty term
λ; γ	Penalty strength; A hyperparameter in MCP
B	Number of bits in quantized weights in APOLLO-OPM
T	Number of cycles in the measurement window, $T \geq 1$
τ	Selected fixed interval size for multi-cycle APOLLO$^\tau$
$x^\tau(k)$	Toggling of proxies at each interval k, $x^\tau(k) \in \mathbb{R}^Q$
$y^\tau(k)$	Power label at each interval k with τ cycles, $y^\tau(k) \in \mathbb{R}$

Table 2: Description of frequently used symbols.

exclusively targets Intel Pentium® processors, [38] is exclusively aimed at the Qualcomm Hexagon 680 DSP, and both works of [58] and [68] target ARM Cortex-A7 and Cortex-A15 processors. Moreover, counter-based power monitors monitor micro-architectural events that manifest several cycles after the causal trigger event. Therefore, they are poorly correlated with recent pipeline activity and are therefore restricted to coarse-grained temporal resolutions.

Compared to counter-based techniques, proxy-based power monitors are much more friendly to automation and applicable to multiple designs [23, 51, 53, 80, 81]. Existing proxy-based techniques suffer from the conflict between low silicon-area overhead and fine-grained temporal resolution. Some of them [23, 51, 53] are coarse-grained with the temporal resolution of thousands of cycles. Their area overhead ranges from 1.5% to 20% over the baseline.

Recent methods [80, 81] improve temporal resolution to 100 cycles. To limit extra overhead from improved resolution, they restrict proxies mostly to primary I/O signals of design modules at selected hierarchy level, significantly reducing the freedom of proxy selection and the underlying power model. Even with this restriction, their area overhead is still > 4% [80, 81]. In [37], a manually-designed digital power meter technique is introduced to address voltage-droop in DSP engines. This technique takes advantage of predictable dataflow patterns that are not available for general-purpose CPUs. In [69], the authors describe a voltage-noise mitigation strategy that combines power proxies with critical path monitors. The work does not formally describe the creation or the accuracy of power proxies in detail. Further, it is unclear whether the methodology is easily portable across designs.

2.3 Position of APOLLO

Although proxy-based techniques have been intensively studied, APOLLO distinguishes itself from previous methods by a new proxy selection technique based on the MCP algorithm. Different from other fully-automatic signal selection methods [23, 40, 44, 51, 53, 72, 80, 81], the selection technique in APOLLO allows flexible selection from any combination of signals (unlike [23, 44, 80, 81]), performs

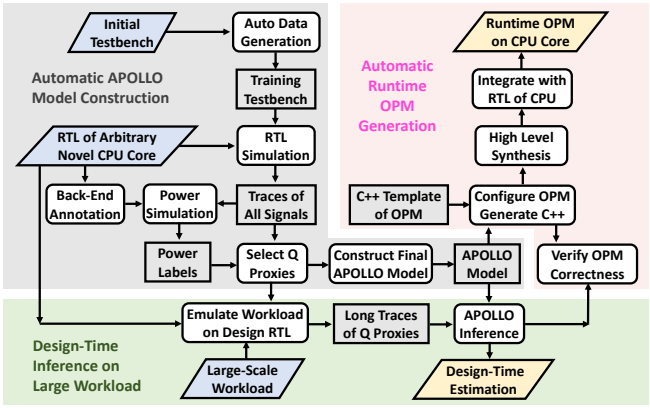


Figure 2: The automated APOLLO framework.

supervised selection (unlike [40]), reduces correlations between proxies (unlike [53]), and proves to be scalable to a large number of candidate signals (unlike [51, 72]). In contrast to [69], the APOLLO framework is a fully automated framework that simultaneously achieves accurate power estimation with per-cycle temporal resolution and generates low-cost silicon implementation with < 1% power/area overhead. This is not obviously achievable by published previous works. Furthermore, APOLLO is proven on commercial million-gate CPUs (Neoverse N1 and Cortex-A77), thus indicating scalability to real-world applications.

3 OVERVIEW OF APOLLO

APOLLO targets the high model accuracy with low computation and implementation cost by automatically identifying representative RTL proxies from a large number of highly-correlated RTL signals and building a lightweight power model. Given a design with M RTL signals S_M, APOLLO selects a subset $S_Q \subset S_M$ with $Q = \lvert S_Q\rvert \ll M$, as power proxies, and Q is the number of proxies. Then it builds a linear power estimator based on S_Q. For per-cycle power tracing,

$$p[i] = \sum_{j=1}^{Q} w_j \cdot x_j[i] \text{ for the } i^{th} \text{ clock cycle,} \tag{1}$$

where $x_1[i], x_2[i], ..., x_Q[i] \in \{0,1\}$ are input features indicating the togglings or transitions of Q proxies in the i^{th} clock cycle, $w_1, w_2, ..., w_Q \in \mathbb{R}^+$ are trainable weights, and $p[i]$ is the predicted power of the same cycle.

Selecting power proxies S_Q from S_M with $Q \ll M$ can greatly accelerate power simulation, reduce data volume for emulation-assisted power analysis, and lower hardware cost for runtime OPM. The choice of Q controls the trade-off between accuracy and efficiency. Although linear power models have been widely used in the past, our proxy selection technique distinguishes APOLLO from previous methods.

Given N cycles of simulation traces, the N ground-truth labels $y[1], y[2], ..., y[N] \in \mathbb{R}^+$ are per-cycle power values generated from a commercial RTL power analysis flow [8], where back-end parasitics are annotated to the RTL design but netlist-level details are abstracted out for flow acceleration in our experiment. It shall be noted that APOLLO applies to an arbitrary method of ground-truth power data collection.

Figure 2 shows the overall APOLLO framework, which can be used as an automated tool with the RTL of an arbitrary CPU design as the input. There are three major components. It firstly constructs the APOLLO model based on automatically generated training data and corresponding power simulation results. Then for design-time estimations, with the assistance of emulation platforms [7], the trained model performs per-cycle inference on large-scale benchmarks. For runtime power monitoring, generic high-level synthesis (HLS) templates are developed in C++ for automatic OPM implementation. The OPM is easily configured based on the model and synthesized together with the microprocessor by EDA tools.

4 APOLLO METHODOLOGY

The total power consumption in a CMOS circuit is contributed by switching and leakage components. Leakage power is determined by the junction temperature and the threshold voltage of transistors. Since it is relatively invariant to code-execution, leakage power measurement is generally not relevant to runtime power management. Similarly, leakage power can be easily estimated using EDA tools [8] at design-time. Therefore, in APOLLO, we focus on modeling the switching component of the total power.

The switching component can be further broken down into dynamic power due to code-dependent charging/discharging of gate/wire-capacitance, short-circuit power during slow signal slews, and glitch power. In practice, power due to glitches and the short-circuit power is much smaller than dynamic power [40], and all three components correlate with signal transitions. The dynamic power at each cycle i can be measured by the summation of power consumed at the capacitance of all toggling gates and wires as

$$Power_{dyn}[i] = \frac{1}{2}V^2 \sum_{g \in \{toggling\ gates\}} C_g \quad (2)$$

Equation (2) does not include the "frequency" component since it is expressed in per-cycle terms. While this approach has signoff-level accuracy, it is computationally intensive and does not scale to workload-execution timescales on large designs with fully annotated parasitics. Since toggling activities of gates are highly correlated with each other, Equation (2) can be reasonably approximated by a simpler linear model based on Q selected proxies as

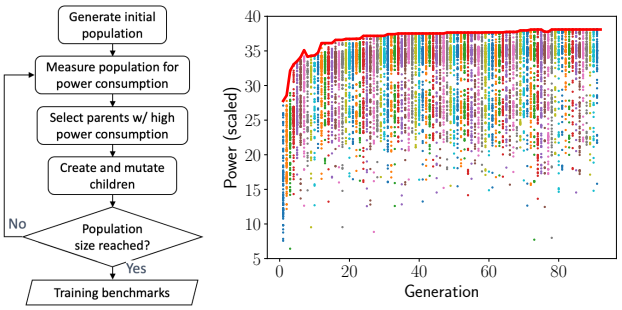
$$Power_{dyn}[i] \approx \frac{1}{2}V^2 \sum_{j \in \{Q\ power\ proxies\}} w_j^{unscaled} \quad (3)$$

Note that the equation in (3) is a measure of the *power-demanded* by the CPU from the power-delivery network (PDN) *before* it is modulated by the PDN response. Hence, the voltage can be viewed as a constant, and by scaling the weights, we reach the simpler final model as Equation (1). In equivalent terms, equation (3) can also be viewed as a measure of the CPU current demand.

4.1 Automatic Training Data Generation

The APOLLO framework starts with automatic training data generation for the target design as shown in Figure 3(a). Generated micro-benchmarks are then replayed with EDA tools to generate power labels.

Previous automatic proxy selection methods [40, 53, 75, 78, 79] mainly adopt three categories of training data: 1) random stimuli, 2) realistic workloads, 3) handcrafted ISA tests or micro-benchmarks.



(a) (b)

Figure 3: Training data generation. (a) GA-based generation flow. (b) A diverse set of training micro-benchmarks with a wide range of measured power.

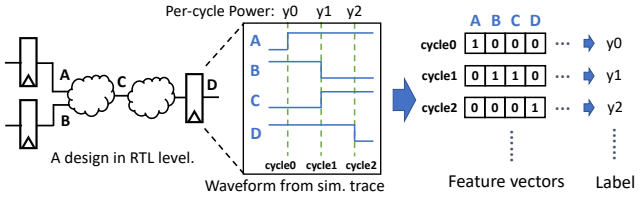
However, for 1), previous studies lack details on how to automatically generate a large number of random stimuli with sufficient diversity for an arbitrary design. For 2), realistic workloads typically include redundant patterns and cannot efficiently cover the full range of power consumption, especially high power consumption scenarios. For 3), it is particularly challenging to generate a diverse training set using manually-developed power benchmarks, even with expert micro-architectural knowledge.

We circumvent these practical engineering challenges by auto-generating the training set of micro-benchmarks using a genetic algorithm (GA)-based framework [28] that is micro-architecture agnostic. Our benchmark-generation flow starts with an initial population of randomly generated micro-benchmarks (referred to as "individual") created with a constrained set of instructions. The average power of each micro-benchmark is then measured using the EDA tool [8]. The ones with highest power are selected as so-called "parents" that are then paired together (crossover) and mutated to create "child" instruction sequences for the next generation and so on. The power measurements of all generated micro-benchmarks are shown in Figure 3(b) across multiple generations. The GA-based optimization loop is primed to generate the worst-case power-consuming benchmark, or a power-virus, as indicated by the envelope of the scatter plot.

As the optimization converges to the worst-case power virus, successive generations favor higher power-consuming benchmarks. However, early generations naturally favor those lower power-consuming benchmarks as the algorithm is yet to identify higher power-consuming instruction sequences. A combination of low and high power-consuming benchmarks across generations naturally creates a rich diversity of benchmarks spanning a large range (>5× ratio between the maximum and minimum individuals) of power consumption.

4.2 Features and Labels Collection

Figure 4 shows the procedure to construct features from the RTL-simulation traces and labels from power simulation results. As Equation (2) shows, per-cycle toggling activities reflect the net transitions and directly correlate with power consumption. At each cycle, for each RTL signal, either a rising or falling edge in the simulation trace is set to 1 as features, while no toggling is set to 0.



(a) Design RTL and simulation trace (b) Feature and label

Figure 4: Feature and label collection based on M RTL signals and N cycles of simulation traces.

As such, each RTL signal contributes to one element in the feature vector.

For M RTL signals and N cycles of simulation traces, the raw input feature vectors are $\mathbf{x} \in \{0,1\}^{N\times M}$, and the input vectors with only Q selected proxies are denoted as $x \in \{0,1\}^{N\times Q}$. The corresponding label is N per-cycle power consumptions $y \in \mathbb{R}^N$ simulated with the EDA tool [8].

4.3 ML-Based Power Proxy Selection

Once raw features and labels are collected, we go through the steps in Figure 5(a) to construct the APOLLO model. The key step is to select Q representative power proxies. It starts with building a temporary linear power model $p' = \sum_{j=1}^{M} w'_j \cdot \mathbf{x}_j$ with all M RTL signals in raw input features. This linear model is not trained only to minimize the prediction error in the training dataset. Instead, when minimizing the prediction error during training, the model simultaneously shrinks all weights $w'_1, w'_2, ..., w'_M$ so that the majority of weights eventually become zero, i.e., the model becomes sparse. Then only those RTL signals associated with non-zero weight terms are selected as power proxies. This procedure is also referred to as pruning. Such sparsity-inducing training is realized by applying a penalty term $\mathcal{P}$ in the loss function to penalize weights. Equation (4) shows the loss function, which consists of both the ordinary prediction error ($\mathcal{L}$) measured in mean squared error and the penalty term ($\mathcal{P}$).

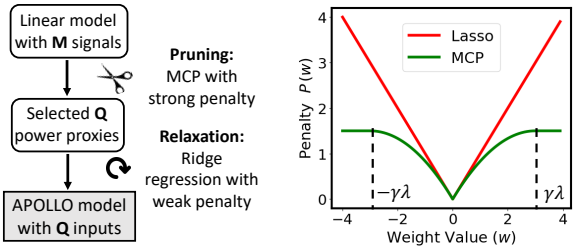
$$\text{Loss} = \mathcal{L} + \mathcal{P} = \frac{1}{N}\sum_{i=1}^{N}(y[i] - p'[i])^2 + \sum_{j=1}^{M} P_{penalty}(w'_j) \quad (4)$$

The sparse linear model is constructed by adopting sparsity-inducing penalty terms. The most widely adopted penalty term for sparsity is Lasso [67], defined as

$$P_{Lasso}(w'_j) = \lambda |w'_j| \quad (5)$$

This Lasso penalty shrinks all weights at the same rate decided by the hyper-parameter λ, which is the penalty strength. However, to ensure $Q \ll M$, we need to set a very large penalty strength λ such that the majorities of weights shrink to zero. As a result, when most small weights shrink to zero and their associated terms are pruned out, the weights of remaining terms are penalized too much to provide accurate power predictions. Based on such an inaccurate model, the selected power proxies are not representative enough.

To overcome the aforementioned limitation, APOLLO adopts the MCP [77] as the penalty term, which is defined by



(a) (b)

Figure 5: APOLLO model construction. (a) Model construction process. (b) Penalty terms of MCP and Lasso.

$$P_{MCP}(w'_j, \gamma > 1) = \begin{cases} \lambda|w'_j| - \frac{w'^2_j}{2\gamma} & \text{if } |w'_j| \le \gamma\lambda \\ \frac{1}{2}\gamma\lambda^2 & \text{if } |w'_j| > \gamma\lambda \end{cases} \quad (6)$$

The hyper-parameter γ in MCP sets a threshold ($\gamma\lambda$) between large and small weights. Figure 5(b) visualizes both P_{Lasso} and P_{MCP} with $\lambda = 1$ and $\gamma = 3$. The absolute derivative of a penalty term indicates the weight shrinking rate during training [54]. Since $|\partial P_{Lasso}/\partial w'_j| = \lambda$, all weights shrink at the same rate λ in Lasso. In comparison, the absolute derivative of MCP penalty is given by

$$\left|\frac{\partial P_{MCP}(w'_j, \gamma > 1)}{\partial w'_j}\right| = \begin{cases} \lambda - \frac{|w'_j|}{\gamma} & \text{if } |w'_j| \le \gamma\lambda \\ 0 & \text{if } |w'_j| > \gamma\lambda \end{cases} \quad (7)$$

Compared with the uniform shrinking rate for P_{Lasso}, large weights with values $> \gamma\lambda$ in MCP do not shrink at all, since derivatives of their penalty terms are zero. For weights with values $< \gamma\lambda$, smaller weights shrink faster. As such, MCP leaves large weights unpenalized and thereby benefits the prediction accuracy of the generated power model. In our experiment, this MCP-based model is efficiently optimized by adopting the coordinate descent method [71] and the proximity operator of MCP [70]. The penalty strength λ can be adjusted to control the number of selected proxies Q.

4.4 Final Model Construction

After power proxy selection by pruning with MCP, we have trained a temporary model $p' = \sum_{j=1}^{M} w'_j \cdot \mathbf{x}_j$ with Q selected proxies S_Q and corresponding non-zero weight terms w'_j. This temporary model can already provide rather accurate predictions. However, although MCP protects larger weights, many remaining weights are still penalized by the large penalty strength λ to a certain extent. To further boost the model accuracy, we train a new linear model $p = \sum_{j=1}^{Q} w_j \cdot x_j$ from scratch with only selected power proxies S_Q. In this new linear model, the ordinary L2 penalty, i.e., ridge penalty [31], is applied, with a much weaker penalty strength compared with the λ used in the previous proxy selection step. This weak ridge penalty is applied to reduce overfitting.

As shown in Figure 5(a), this step is named *relaxation* and generates the final APOLLO power model. During the previous proxy selection step, to shrink most weights to zero, the penalty term $\mathcal{P}$ dominates the loss, and the prediction error $\mathcal{L}$ is less optimized. This *relaxation* can be viewed as a fine-tuning stage to better optimize $\mathcal{L}$. Since L2 is not sparsity-inducing, the number of proxies Q remains unchanged.

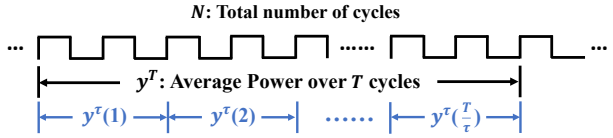


Figure 6: Multi-cycle APOLLO model — Power label is y^T, with a measurement window size of T cycles. Label at each interval is y^τ, with selected interval of τ cycles.

4.5 Multi-Cycle Power Modeling

In previous subsections, we construct the APOLLO model for per-cycle power tracing. As we show in Section 8, fine-grained temporal resolution enables applications like voltage droop mitigation. In this sub-section, we generalize the APOLLO model to larger time-window sizes. Like Figure 6 shows, this multi-cycle model estimates the average power over a time window with T cycles, which is chosen to be a power of two for ease of efficient hardware implementation.

A straightforward multi-cycle solution is to directly use the average of T per-cycle power predictions p^T over the T-cycle window[2]. It uses the same per-cycle model for any T. Such an approach captures details of individual clock cycles but neglects correlations among different clock cycles. Alternatively, one can average the transitions over T cycles and generate a T-cycle power estimation based on the average toggling rate. However, this approach loses useful information such as cycle-details that can be particularly helpful when T becomes large. In addition, the model developed by this approach is dependent on the varying T. In Section 7.3, we show that both the average-prediction and the average-input approaches fail to provide an accurate and robust solution.

We introduce a multi-cycle estimation technique that overcomes the weakness of the aforementioned approaches. A time window of T is divided into multiple intervals of τ cycles. The values of T and τ are selected such that T is integer multiples of τ. An example is shown in Figure 6. During the model construction and training, for each τ-cycle interval k, we measure both the average toggling activities $\mathbf{x}_1^\tau(k), ..., \mathbf{x}_M^\tau(k) \in \mathbb{R}$ and the average power $y^\tau(k) \in \mathbb{R}$ over the τ cycles[3]. Based on these raw inputs and labels, we execute the same training procedure as the per-cycle model to select Q power proxies S_Q with features $x_1^\tau, ..., x_Q^\tau$. The result is a τ-cycle model denoted as APOLLO$^\tau$, whose weights are denoted as $\omega_1, ..., \omega_Q$. It is to be noted that the construction of APOLLO$^\tau$ is independent of T, and its performance is controlled by selecting an appropriate τ value as a hyper-parameter before training.

At the inference stage, there are $\frac{T}{\tau}$ intervals in a time window. As Figure 6 shows, the final prediction $p^T \in \mathbb{R}$ at each T-cycle window is the average over these $\frac{T}{\tau}$ predictions from the APOLLO$^\tau$ model:

$$p^T = \frac{1}{T/\tau} \sum_{k=1}^{T/\tau} p^\tau(k) \text{ , where } p^\tau(k) = \sum_{j=1}^{Q} \omega_j \cdot x_j^\tau(k) \qquad (8)$$

[2]We use the superscript on a variable to denote the average of the variable over a timing window with multiple cycles.
[3]We use parentheses and brackets to differentiate the indices of intervals and cycles.

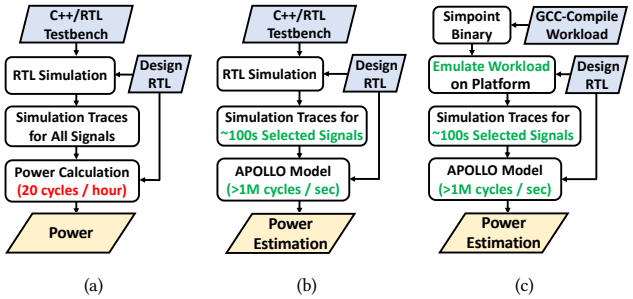


Figure 7: Design-time per-cycle power analysis flows. (a) Commercial power analysis. (b) APOLLO-based power analysis. (c) APOLLO integrated with emulator-assisted power analysis for large-scale benchmarks.

Here, the input $x_j^\tau(k) \in \mathbb{R}$ for each interval is a real number instead of a binary. If directly implemented on hardware, this requires Q counters and multipliers like previous OPMs [23, 51, 80, 81]. In contrast, the toggling in each cycle is a binary number and thus the per-cycle model can be implemented by AND gates instead of multipliers. To avoid multipliers for on-chip implementation of the multi-cycle model, we rearrange the inference process in Equation (8) as below:

Take the first interval $p^\tau(1)$ when $k = 1$ as example:

$$p^\tau(1) = \sum_{j=1}^{Q} \omega_j \cdot x_j^\tau(1) = \sum_{j=1}^{Q} \omega_j \cdot \frac{1}{\tau} \sum_{i=1}^{\tau} x_j[i] = \frac{1}{\tau} \sum_{i=1}^{\tau} \sum_{j=1}^{Q} \omega_j \cdot x_j[i]$$

$$\text{Thus, } \quad p^T = \frac{1}{T/\tau} \sum_{k=1}^{T/\tau} p^\tau(k) = \frac{1}{T} \sum_{i=1}^{T} \sum_{j=1}^{Q} \omega_j \cdot x_j[i] \qquad (9)$$

In Equation (9), the weights are multiplied with binary numbers instead of real numbers. This new inference process can be regarded as predicting T-cycle average power according to per-cycle toggles. As such, it takes per-cycle details, considers correlations among multiple cycles, and hence overcomes the drawbacks of aforementioned approaches. Interestingly, τ is no longer needed in inference. By setting T to be power of 2, the division in Equation (9) can be realized by directly discarding the $\log_2(T)$ lowest bits. Therefore, the on-chip implementation of this multi-cycle model can reach low hardware overhead like the per-cycle model.

5 DESIGN-TIME POWER ANALYSIS

A typical conventional design-time power analysis flow is shown in Figure 7(a). It generates simulation traces for all signals in VCD or FSDB file format through RTL simulation, then performs power calculation with simulation tools using these traces. Such a flow is very time-consuming. One major bottleneck is the last step of power calculation, which is extremely slow for large designs.

To accelerate this process, we incorporate the APOLLO model into the flow as shown in Figure 7(b), where the number of signals to be traced is greatly reduced and the last step of power calculation is replaced by APOLLO. APOLLO can infer power for millions of cycles within seconds. This APOLLO-assisted power simulation flow works well for cases where RTL simulation time is reasonable.

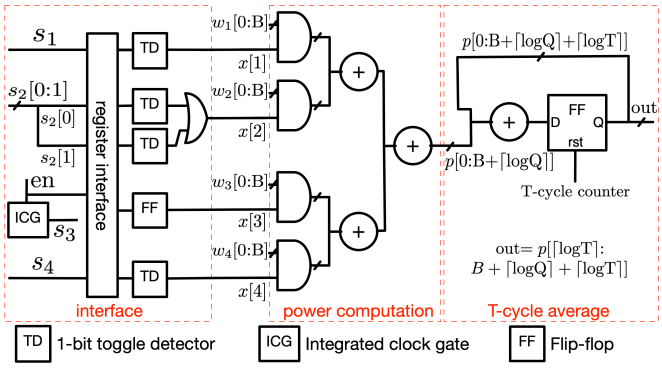


Figure 8: OPM integration with the CPU design.

For long-running benchmarks, the RTL simulation step in Figure 7(b) becomes an execution bottleneck. We propose to overcome this using an emulator-assisted power analysis flow as shown in Figure 7(c). In this flow, millions of benchmark cycles are emulated on a commercial platform [7] with dedicated hardware to generate million-cycle simulation traces in minutes. In the absence of APOLLO, the power-simulation flow requires switching details of all nets to be dumped. For a large industrial-scale design, this can easily exceed hundreds of GB leading to storage and memory capacity issues during power analysis. Traditionally, this problem is circumvented by estimating power at a coarse-grained temporal resolution, e.g., thousands of clock cycles. With APOLLO, the data is reduced by several orders of magnitude by only collecting the toggling activities of Q power proxies, and a cycle-accurate estimation for the emulator-assisted flow is enabled.

6 RUNTIME ON-CHIP POWER METER

APOLLO provides an accurate and fine-resolution runtime OPM with low hardware cost. The OPM implements a linear model with Q power proxies as the input, which is a binary vector at each cycle, e.g., $x[i] \in \{0,1\}^Q$. All weights are quantized into B-bit fixed-point values, which can be configured to accommodate potential model re-training using sign-off or hardware measurement power values.

The APOLLO-OPM is fully integrated with the microprocessor. Figure 8 shows the OPM consists of three components, i.e., "interface", "power computation" and "T-cycle average". The "interface" latches the input signals using the register interface and then extracts per-cycle toggling activities as single-bit values for each power proxy. The register interface minimizes the data path timing impact from OPM on the original design. The "interface" takes power proxies, i.e., s_1, $s_2[0:1]$, s_3 and s_4 as inputs, which are further categorized into three cases: (1) 1-bit signal (s_1 and s_4). A 1-bit toggle detector "XOR"s the monitored signal with its registered version to determine whether a toggle occurred. (2) Bus signal ($s_2[0:1]$). We set up 1-bit signal interface for each bit of the bus. An extra OR gate determines whether the entire bus signal toggles. (3) Gated clock signal (s_3). A gated clock signal (s_3) toggles twice during one clock cycle. Instead of using a 1-bit toggle detector, we automatically trace the clock enable signal (*en*), which is directly latched using a flip-flop to determine whether gated clock signal toggles at the same cycle as other power proxies.

Methods	[75]	[40]	[23, 51] [80, 81]	[53]	APOLLO	
					Per-cycle	Multi-cycle
#Counter	0	Q	Q	Q	1	1
#Multiplier	$\propto M$	$\propto Q^2$	Q	1	0	0

Table 3: Hardware implementations of runtime monitors or design-time emulators with Q selected proxies.

The "power computation" component calculates the intermediate values from the quantized weights, i.e., $w_j[0:B]$, and per-cycle toggling values, i.e., $x[j]$. The bit width of these power values is extended to $B+\lceil logQ \rceil$ to ensure the full precision addition. After intermediate values are computed on a cycle-by-cycle basis, a "T-cycle average" component computes the average power over T cycles using flip-flops and adders. The flip-flop reset is controlled by a T-cycle counter, which resets the value of output, i.e., *out*, every T cycles. Similarly, the bit width of intermediate values is extended to $B+\lceil logQ \rceil+\lceil logT \rceil$ to guarantee full precision addition. The output power value needs to be divided by T according to Equation (9). This is realized by dropping the lowest $\log\lceil T \rceil$ bits as T is set to be the power of 2.

The OPM structure in Figure 8 is applicable to both per-cycle and multi-cycle power model, due to the linear model structure discussed in Equation(9). The OPM is implemented with generic templates (configurable in B, Q and T) in C++ using the Catapult HLS tool [5] and synthesized into gate-level netlist using Design Compiler [6]. Section 7.5 further explores the trade-off between the OPM's accuracy and gate area by varying Q and the bitwidth B.

The key to the low-cost implementation is two-fold. First, the APOLLO only selects $< 0.05\%$ RTL signals as power proxies. Secondly, calculation of the per-cycle power only requires a conditional accumulation of the proxy weights depending upon whether they toggled or not. As such, only a set of AND gates and adders, instead of multipliers, are needed for the computation. Table 3 shows a comparison between APOLLO-OPM and those in previous studies. The hardware implementation cost of the APOLLO model is much lower than previous approaches, such as Simmani [40]. Most previous OPMs require a counter and multiplier for each proxy, which incurs a much larger area cost. Furthermore, although APOLLO-OPM may include different sets of trained weights from per-cycle and multi-cycle power model, they share the same hardware structure, which allows greater flexibility and configurability compared to previous studies.

7 EXPERIMENTAL RESULTS

7.1 Data Generation and Experiment Setup

In our experiments, micro-benchmarks used in model training and testing are kept strictly different and separate. Through the automatic training data generation, $> 1,000$ random micro-benchmarks are obtained in 4 days to cover a wide range of average power consumption, among which around 300 micro-benchmarks are selected to form the training set with a uniform power distribution. 20% of the training data are selected to form a validation set for parameter tuning. Unlike the training data, which are automatically generated, the testing data are from 12 representative micro-benchmarks handcrafted by CPU designers corresponding to various use cases, as shown in Table 4. They cover both low- and

Name	dhrystone	maxpwr_cpu	dcache_miss	saxpy_simd
Cycles	1222	600	654	1986
Name	maxpwr_l2	icache_miss	cache_miss	daxpy
Cycles	1568	800	600	1600
Name	memcpy_l2	throttling_1	throttling_2	throttling_3
Cycles	3000	1100	1100	1100

Table 4: Designer-handcrafted testing benchmarks.

high-power consumption regions. The three micro-benchmarks named 'throttling' reflect applying different throttling schemes [3] to the microprocessor. The simulation trace lengths N for training and testing are approximately 30,000 and 15,000 cycles on Neoverse N1, respectively.

All experiments are firstly performed on the Neoverse N1 [21, 57], a microprocessor for a wide range of cloud-native server workloads executing at world-class performance and efficiency. To verify the robustness of APOLLO on different designs, we further test on Cortex-A77 [2], a high-performance energy-efficient microprocessor targeting mobile and laptop devices. 5,000 cycles of training data and 2,000 cycles of testing data are generated for Cortex-A77. The numbers of RTL signals M are $> 5 \times 10^5$ and $> 1 \times 10^6$ for Neoverse N1 and Cortex-A77, respectively.

The RTL simulation is performed using VCS® [9] and the ground-truth power is simulated by PowerPro® [8] based on a commercial 7nm technology setup. All ML models are implemented with Python v3.7. For baseline methods, CNN-based models are based on Pytorch v1.5 [55], and other models are implemented with scikit-learn v0.22 [56]. For APOLLO, we implement the MCP algorithm and the coordinate descent algorithm with NumPy [30]. During training, the MCP regressor converges within 200 iterations, with the threshold of unpenalized weights set to $\gamma = 10$. The overall proxy selection and model training time of APOLLO and all baseline methods are within three hours, which is affordable.

All accuracies are measured on the testing data. Metrics include the coefficient of determination (R^2) [50], the normalized root mean squared error (NRMSE), and the normalized mean absolute error (NMAE), defined as follows. The $\bar{y}$ is the average over all N labels $y[i]$.

$$\text{NRMSE} = \frac{1}{\bar{y}}\sqrt{\frac{\sum_{i=1}^{N}(y[i]-p[i])^2}{N}},\ \text{NMAE} = \frac{\sum_{i=1}^{N}|y[i]-p[i]|}{\sum_{i=1}^{N} y[i]}$$

7.2 Baseline Methods

In experimental comparisons, it is difficult to exhaust the significant body of previous researches for various target designs and application scenarios. Our solution is to compare the accuracy of APOLLO with representative approaches that target the highest accuracy with a high level of acceptable computation complexity. These complex non-linear methods [40, 79] prove to outperform simple linear models adopted in most runtime approaches. We also compare with a recent runtime technique [53] which uses a sparsity-induced algorithm. Table 5 shows comparisons with Simmani [40], PRIMAL [79], and Pagliari et al. [53]. For Simmani, signals are clustered with K-means algorithm and power proxies are selected from

Works	Simmani [40]	PRIMAL [79]	PCA [79]	Lasso [53]	APOLLO
Proxies Selection	K-means	✗	✗	Lasso	MCP
Pre-Processing	Polynomial	✗	PCA	✗	✗
ML Model	Elastic Net	CNN	Linear	Linear	Ridge

Table 5: Comparisons with baseline methods.

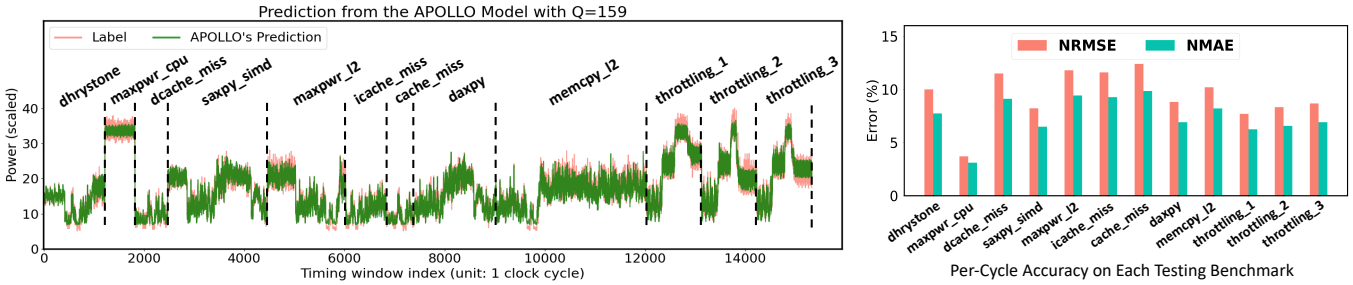
different clusters. After that, toggling activities of both the Q power proxies and the Q^2 2^{nd} order polynomial terms are adopted as potential model features. The adopted elastic net model is a linear model with a combination of both Lasso and Ridge penalties, where the power measurement window size T is a hyperparameter tuned to improve model accuracy.

PRIMAL [79] targets accurate design-time simulation on software with several methods, among which the CNN produces best results and is adopted for comparison. It uses all flip-flop signals as input proxies without any selection. As the number of flip-flops is at least one order of magnitude greater than typical values of Q, the simulation/emulation cost of PRIMAL is much higher than APOLLO. Moreover, the use of CNN makes it impractical for runtime OPM. Another method proposed by PRIMAL [79] is principal components analysis (PCA). It shall be noted that dimension reduction techniques like PCA still require the toggling activities of all candidate signals as the initial input during inference. This is computationally expensive and fundamentally different from proxy selections. Pagliari et al. [53] adopt Lasso regression, the most widely-used sparsity-inducing method, for proxy selection and model construction. For previous methods considering only flip-flop signals as input features, to avoid underestimation of their accuracy, we implement them with all RTL signals as input features for a fair comparison. This is expected to generate better accuracy than limiting proxies only to flip-flop signals.

7.3 Accuracy of APOLLO

For per-cycle power estimation, APOLLO is compared with other methods in Figure 10, which measures the trade-off between Q and corresponding prediction accuracy on Neoverse N1. The previous Lasso-based method [53] and Simmani [40] are also applied to the per-cycle estimation for a fair comparison. Both CNN in PRIMAL and the PCA model are represented by horizontal lines since their $Q = M$ in this comparison. APOLLO achieves NRMSE $< 10\%$ and $R^2 > 0.95$ with $Q \approx 150$, which is less than 0.03% of total RTL signals in Neoverse N1. It shows similar NRMSE when comparing PRIMAL with APOLLO at $Q = 500$. In contrast, the NRMSE of Simmani and Lasso is higher than 12% even with $Q = 500$. This explains why the previous Lasso-based method [53] and Simmani [40] restrict their applications to coarse-grained temporal resolution.

We provide a detailed evaluation of the APOLLO model with $Q = 159$, which obtains NRMSE $= 9.4\%$ and $R^2 = 0.95$. Figure 9(a) illustrates prediction p and label y as power traces on the 15,000-cycle testing dataset, covering all 12 handcrafted micro-benchmarks. APOLLO's prediction overlaps well with the ground truth for distinctive patterns from different benchmarks. Figure 9(b) measures the accuracy in NRMSE and NMAE for each individual micro-benchmark. The NMAE is less than 10% for all benchmarks.



(a) (b)

Figure 9: Evaluation of an APOLLO model with $Q = 159$ (Neoverse N1).

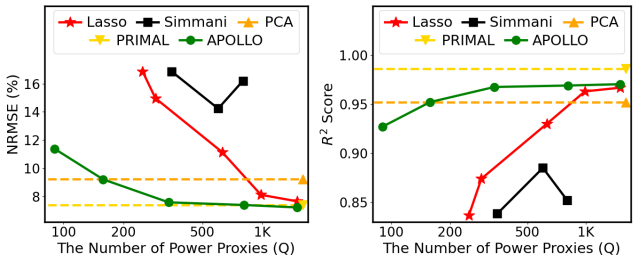


Figure 10: Per-cycle power accuracy vs. number of proxies for per-cycle power prediction (Neoverse N1).

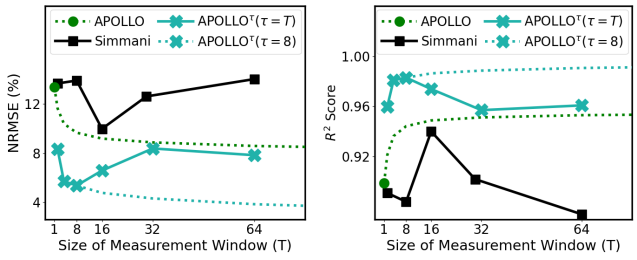


Figure 11: T-cycle accuracy vs. window size (T) for multi-cycle prediction (Neoverse N1). — $Q = 200$ for Simmani, $Q = 70$ for APOLLO methods.

The APOLLO method can enable relative power comparisons across microarchitecture configurations, since it leads to generally unbiased power predictions that neither consistently over-estimate nor under-estimate a microarchitecture. Such unbiased predictions originate from the rich diversity in our automatically generated training data, covering both low- and high-power benchmarks of each design. As Figure 9(a) shows, *averaged* predicted and ground-truth power are close for all testbenches on Neoverse N1. The averaged ground truth is 16.9 and the prediction is 16.8, showing merely 0.6% difference (similar for Cortex-A77). Thus, microarchitectural comparisons can be made easily if the relative difference in the power consumption exceeds this small error bar.

Figure 11 estimates power over measurement windows with T cycles. Previous multi-cycle model Simmani [40] is trained and validated for different T values $\{4, 8, 16, 32, 64\}$. APOLLO in Figure 11 stands for the simple average over T per-cycle predictions. The green dotted line means predictions of various T values are all averaged from the same *per-cycle* APOLLO model. In comparison, several multi-cycle APOLLO$^\tau$ models with interval sizes $\tau = T = \{4, 8, 16, 32, 64\}$ are trained. Results show that $\tau = 8$ provides the best accuracy. We thus choose $\tau = 8$ for multi-cycle model and the dotted line is from APOLLO$^\tau(\tau = 8)$ for all T values. Notice that $Q = 200$ for Simmani, while all APOLLO-based models keep $Q = 70$. In Figure 11, the simple average of per-cycle APOLLO is already more accurate than Simmani for all T values using around one-third of proxies. The multi-cycle APOLLO$^\tau$ with $\tau = 8$ further improves NRMSE by 5%. This supports our claim in Section 4.5, indicating that both simple average of per-cycle model ($\tau = 1$) and directly averaging inputs for any T ($\tau = T$) fails to provide the most accurate and robust solution.

To verify that APOLLO generalizes well on different designs, we measure the per-cycle accuracy on Cortex-A77. The comparisons are shown in Figure 12. Similar to the trend in Figure 10, APOLLO achieves NRMSE = 8% when $Q \approx 300$, which is less than 0.03% of total RTL signals in Cortex-A77, while Simmani and Lasso show NRMSE > 10% with $Q = 500$. In addition, APOLLO obtains comparable NRMSE with the CNN in PRIMAL when $Q = 500$.

7.4 Model Discussion

We provide insights into APOLLO's high-quality predictions from two additional perspectives. First, with the same Q, the MCP adopted by APOLLO allows large weights compared with the Lasso. This is verified in Figure 13, which reports the summation of all Q absolute weights in each model. Second, the correlation among the selected power proxies can jeopardize the generalization of models. Figure 14 shows the average variance inflation factor (VIF) [74], which quantifies the correlation among proxies for each method.

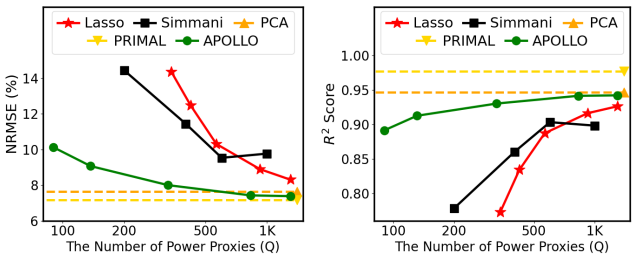


Figure 12: Per-cycle power accuracy vs number of proxies for per-cycle power prediction (Cortex-A77).

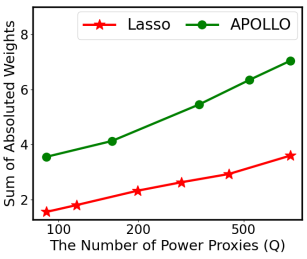


Figure 13: Sum of all absolute weights.

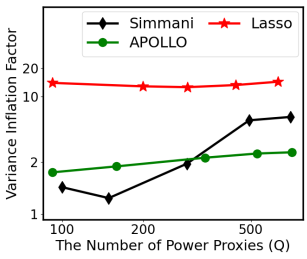


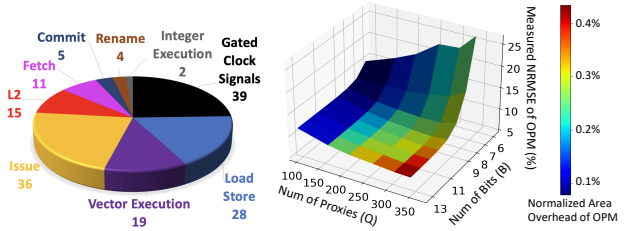
Figure 14: Variance inflation factors (VIF).

APOLLO shows a much lower VIF than Lasso regression. By shrinking weights with different rates, the MCP tends to treat correlated RTL signals differently so that correlated ones are not selected simultaneously as proxies. Another observation is that Simmani also achieves low VIF by selecting power proxies from different clusters. However, since the clustering-based selection is unsupervised, the correlation between power proxies and the label is not as directly optimized as APOLLO. Simmani is not covered in Figure 13 as it is not a linear model and its weights are not comparable with APOLLO/Lasso.

We further categorize the Q APOLLO-extracted proxies based on the RTL signal properties: 1) determine whether a proxy is a gate clock signal; 2) for a non-clock RTL signal, determine which functional unit it belongs to. Figure 15(a) shows the distributions of the 159 power proxies for Neoverse N1 CPU based on the aforementioned RTL signal properties. 39 power proxies are gated clock signals, which means APOLLO captures the major contributor, i.e., clock network, of the dynamic power consumption. Furthermore, with the APOLLO model, the weights of the gated clock signals provide useful insights into the power-hungry clock gating structure, which sets guidelines for designers to further optimize clock power. APOLLO model also captures significant power contributors, such as "Vector Execution" (19 out of 159), "Issue" (36 out of 159), and "Load Store" (28 out 159). These power proxies are critical indicators to enhance the throttling schemes and mitigate CPU maximum power consumption [3].

7.5 Hardware Prototype of APOLLO-OPM

We synthesize the APOLLO model as an OPM under the same target frequency and 7nm technology as Neoverse N1 CPU. The model accuracy is measured in NRMSE and the cost is quantified by area overhead. The trade-off between accuracy and area normalized by the total gate area of Neoverse N1 is shown in Figure 15(b). By varying the number of selected proxies Q and the number of bits B used for weight quantization, such trade-off curve is explored to help determine appropriate values for Q and B. Although we are exploring the area and accuracy trade-off using a per-cycle power model, our automated OPM generation accommodates the average power computation over T cycles and the only extra hardware cost is one $B + \lceil \log Q \rceil + \lceil \log T \rceil$-bit flip flop and adder. To evaluate the accuracy of this implementation, we simulate our hardware solution with the 15,000-cycle testing data of Neoverse N1. According to Figure 15(b), both Q and B have a considerable impact on accuracy and area. For all Q values, the accuracy loss is high for $B < 9$ and becomes negligible when $B > 10$. Thus, our strategy is to keep



(a) (b)

Figure 15: (a) Distribution of extracted power proxies from Neoverse N1 microprocessor. (b) Trade-off between the area overhead and accuracy (NRMSE) of the OPM.

$B \approx 10$ and vary Q to generate different solutions. Specifically, with 10-bit weights, the quantization leads to $< 0.1\%$ NRMSE increase compared with the APOLLO model on software at design-time. For an OPM with $B = 10$ and $Q = 159$, its total gate area is only 0.2% of the gate area of Neoverse N1. It has a latency of 2 cycles.

OPM overheads are analyzed using physical implementation estimations with the overall Neoverse N1 CPU, for the OPM placement region at a central location within the CPU floorplan, bounded as illustrated in Figure 1. Individual proxies routed from different blocks to the centralized OPM require buffering that incurs area and power overheads. On the Neoverse N1 CPU, we budget a single clock cycle to account for the latency of routing multiple proxies to the OPM by registering all inputs at the OPM interface (Figure 8), at the expense of an extra cycle latency.

Driving the proxies to the centralized OPM requires high-strength buffers that contribute an additional 0.4% power overhead. The OPM circuitry itself consumes 0.5% power overhead, leading to an overall power overhead of 0.9% compared to the baseline CPU power at 3GHz in a commercial 7nm technology. In comparison, the reported power overheads of all previous proxy-based runtime monitors are $1.9 - 14\%$ [23], $2.7 - 4\%$ [51], 5.7% [53], 10% [80], and 4.7% [81]. The total area overhead remains negligible ($< 0.5\%$).

8 APPLICATION SCENARIOS

The accuracy of APOLLO model in estimating the power leads to new applications for power management in high-performance CPUs, during both design-time and runtime.

8.1 Design-Time Power Introspection

We described in Section 5 how the APOLLO model can be integrated into an emulator-assisted workload simulation framework. By only recording the toggle trace of $Q = 150$ power proxies, the size of a simulation trace with $N = 17$ million cycles on Neoverse N1 is reduced to only 1.1 GB. The entire trace is generated on Palladium® Z1 emulation platform [7] within 3 minutes. This capability enables accurate generation of power trace spanning >10M processor cycles within minutes, enabling unprecedented design-time power introspection. Figure 16 illustrates this in the power trace generated for the "hmmer" benchmark from the SPEC2006 on the Neoverse N1. We show only a portion (40,000 cycles) of the whole trace to illustrate distinct transitions in the CPU power and current demand.

Achieving this using EDA tools is computationally infeasible for industry-scale CPU designs. We estimate the inference time on one billion cycles, covering 1/3 of a second in chip runtime for

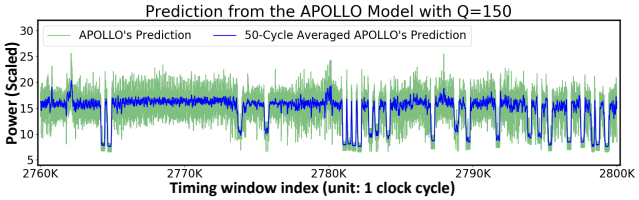


Figure 16: A portion (40,000 over 17 million cycles) of power estimation from the APOLLO-integrated emulator-assisted power analysis (Neoverse N1).

the 3GHz Neoverse N1. With a linear model, APOLLO inference only takes one minute with $Q < 500$. In comparison, the CNN model in PRIMAL takes months and the PCA takes around one week, since both algorithms do not perform proxies selection. As for Simmani, since it takes approximately Q^2 polynomial terms as input, its inference time can increase quadratically with Q. It may take Simmani days for inference of a billion cycles when $Q = 1000$.

8.2 Runtime Proactive Ldi/dt Mitigation

Using the APOLLO-OPM's per-cycle estimation capability, it is possible to predict Ldi/dt voltage-droop events ahead of time before their actual occurrence at a low cost[4]. We intend to develop this further in our future work, but here we provide a brief conceptual description of how this can be realized using the OPM. The differentiation (di/dt) operator in continuous time is equivalent to the differencing (ΔI) in discrete time. We plot both the OPM readings on Neoverse N1 and the ground truth ΔI samples (scaled to arbitrary units) from PowerPro [8] in a scatter plot in Figure 17. The plot is in log scale to cover a wide data range with visibility to details. The Pearson's correlation [46] between OPM and the ground truth reaches 0.946, indicating a high correlation.

The points in the bottom-right and the top-left quadrants indicate samples where OPM estimations depart significantly from the ground truth. The signal magnitudes recorded in these quadrants are near the origin (indicating small-magnitude delta current) as a consequence of the OPM accuracy. Points in the top-right quadrant indicate cycles where there is an increased current demand relative to the previous cycle. Such cycles are typically precursors to voltage-droop events. The bottom-left quadrant indicates a drastic reduction in current demand leading to potential voltage-overshoots. For the samples in deep droop and overshoot regions, APOLLO OPM correlates well with the ground truth. This indicates that the OPM can accurately estimate CPU current transients, and thus enable circuit-level mitigation schemes such as adaptive-clocking to engage prior to the development of voltage-droop.

Proactive droop mitigation using proxies has been proposed in prior art [37, 69]. In [69], authors describe a combination of pipeline event indicators and digital power-proxies for droop-event indication. However, the technique for creating this proxy is not formally described. The work of [37] describes proactive mitigation

[4]In [59], authors describe an online training approach where a voltage-emergency signature is dynamically learned to predict future noise events. This approach requires a checkpoint and recovery mechanism for initial failures when no signature has been learned. This approach is onerous to implement in industrial CPU designs. Correctness in presence of corner cases is difficult to guarantee.

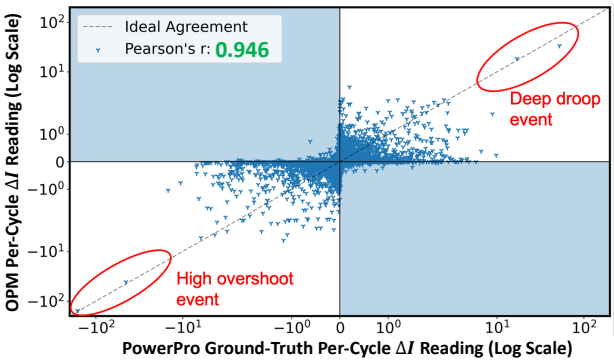


Figure 17: Voltage droop analysis based on per-cycle power on Neoverse N1, showing OPM prediction versus ground-truth (scaled to arbitrary units). Note that axes are in log-scale to visually magnify the uncorrelated samples that are actually very small in magnitude. The Pearson coefficient is 0.946, indicating a high correlation between estimations and ground-truth.

on the Hexagon DSP engine. DSP engines are data-plane dominated, in contrast with CPUs that are control-plane dominated. As such, manual design for CPU power-proxies is significantly harder, particularly when fine-grained temporal resolution is necessary.

9 CONCLUSION AND FUTURE WORK

Power introspection is increasingly important in modern high-performance CPU designs, for both design-time optimization and runtime management. This has particular significance in many-core infrastructure SoCs in ultra-scaled technology nodes. Within a unified framework, APOLLO bridges an important technology gap by providing both cycle-accurate design-time power simulation and low-overhead on-chip power metering. We demonstrate that by monitoring < 0.05% RTL signals, the OPM achieves $R^2 > 0.95$ with <1% area/power overhead when integrated with Neoverse N1.

Our future research is focused on two directions. Firstly, we will further develop and quantify margin reduction using proactive Ldi/dt mitigation with OPM. Secondly, we will focus on translating the APOLLO design-time model into higher abstraction models (C/C++ instead of RTL), thereby integrating performance simulation with power-tracing. Ultimately, the APOLLO capability can enable the development of new mechanisms for smarter power and thermal management in future SoCs. The framework is extensible to diverse compute engines and is therefore a compelling addition to the microarchitects' toolbox.

ACKNOWLEDGMENTS

This work was conducted under the aegis of high-performance A-Class CPU research program at Arm Research. The authors thank Matt Elwood (power architect for multiple generations of Arm CPUs) for his excellent critique and feedback, without which this work would not have been possible. This work is partially supported by NSF-2106828, NSF-2112562 (NSF AI Institute - Athena), and Semiconductor Research Corporation (SRC) Tasks 2810.021 and 2810.022 through UT Dallas' Texas Analog Center of Excellence (TxACE).

REFERENCES

[1] 2021. Ampere Altra SoC. https://amperecomputing.com/altra/.

[2] 2021. Arm Cortex-A77 Core Technical Reference Manual. https://developer.arm.com/documentation/101111/latest/preface.

[3] 2021. Arm Neoverse N1 Core Technical Reference Manual. https://developer.arm.com/documentation/100616/0301.

[4] 2021. AWS Graviton Processor. https://aws.amazon.com/ec2/graviton/.

[5] 2021. Catapult® High-Level Synthesis. https://www.mentor.com/hls-lp/catapult-high-level-synthesis/.

[6] 2021. Design Compiler® RTL Synthesis. https://www.synopsys.com/implementation-and-signoff/rtl-synthesis-test/design-compiler-nxt.html.

[7] 2021. Palladium® Z1 Enterprise Emulation Platform. https://www.cadence.com/en_US/home/tools/system-design-and-verification/acceleration-and-emulation/palladium-z1.html.

[8] 2021. PowerPro® RTL Low-Power. https://www.mentor.com/hls-lp/powerpro-rtl-low-power/.

[9] 2021. VCS® functional verification solution. https://www.synopsys.com/verification/simulation/vcs.html.

[10] Frank Bellosa. 2000. The benefits of event: driven energy accounting in power-sensitive systems. In *ACM SIGOPS European Workshop (EW)*.

[11] Ramon Bertran, Marc Gonzalez, Xavier Martorell, Nacho Navarro, and Eduard Ayguade. 2010. Decomposable and responsive power models for multicore processors using performance counters. In *ACM ICS*.

[12] Srikar Bhagavatula and Byunghoo Jung. 2012. A low power real-time on-chip power sensor in 45-nm SOI. *IEEE Transactions on Circuits and Systems I: Regular Papers (TCAS-I)* (2012).

[13] Srikar Bhagavatula and Byunghoo Jung. 2013. A power sensor with 80ns response time for power management in microprocessors. In *CICC*.

[14] Nathan Binkert, Bradford Beckmann, Gabriel Black, Steven K Reinhardt, Ali Saidi, Arkaprava Basu, Joel Hestness, Derek R Hower, Tushar Krishna, Somayeh Sardashti, et al. 2011. The gem5 simulator. *ACM SIGARCH computer architecture news* (2011).

[15] Nathan L Binkert, Ronald G Dreslinski, Lisa R Hsu, Kevin T Lim, Ali G Saidi, and Steven K Reinhardt. 2006. The M5 simulator: Modeling networked systems. *IEEE Micro* (2006).

[16] W Lloyd Bircher and Lizy K John. 2007. Complete system power estimation: A trickle-down approach based on performance events. In *IEEE ISPASS*.

[17] Alessandro Bogliolo, Luca Benini, and Giovanni De Micheli. 2000. Regression-based RTL power modeling. *ACM TODAES* (2000).

[18] Keith A Bowman, Sarthak Raina, J Todd Bridges, Daniel J Yingling, Hoan H Nguyen, Brad R Appel, Yesh N Kolla, Jihoon Jeong, Francois I Atallah, and David W Hansquine. 2016. A 16 nm All-Digital Auto-Calibrating Adaptive Clock Distribution for Supply Voltage Droop Tolerance Across a Wide Operating Range. *IEEE JSSC* (2016).

[19] David Brooks, Pradip Bose, Viji Srinivasan, Michael K Gschwind, Philip G Emma, and Michael G Rosenfield. 2003. New methodology for early-stage, microarchitecture-level power-performance analysis of microprocessors. *IBM Journal of Research and Development* (2003).

[20] David Brooks, Vivek Tiwari, and Margaret Martonosi. 2000. Wattch: A framework for architectural-level power analysis and optimizations. *ACM SIGARCH Computer Architecture News* (2000).

[21] Robert Christy, Stuart Riches, Sujil Kottekkat, Prasanth Gopinath, Ketan Sawant, Anitha Kona, and Rob Harrison. 2020. A 3GHz ARM Neoverse N1 CPU in 7nm FinFET for Infrastructure Applications. In *ISSCC*.

[22] Joel Coburn, Srivaths Ravi, and Anand Raghunathan. 2005. Power emulation: a new paradigm for power estimation. In *DAC*.

[23] Luca Cremona, William Fornaciari, and Davide Zoni. 2020. Automatic identification and hardware implementation of a resource-constrained power model for embedded systems. *Elsevier Sustainable Computing: Informatics and Systems* (2020).

[24] C Gilberto and M Margaret. 2005. Power prediction for intel xscale processors using performance monitoring unit events. In *ISLPED*.

[25] Waclaw Godycki, Christopher Torng, Ivan Bukreyev, Alyssa Apsel, and Christopher Batten. 2014. Enabling realistic fine-grain voltage scaling with reconfigurable power distribution networks. In *MICRO*.

[26] Bhavishya Goel, Sally A McKee, Roberto Gioiosa, Karan Singh, Major Bhadauria, and Marco Cesati. 2010. Portable, scalable, per-core power estimation for intelligent resource management. In *International Conference on Green Computing (IGCC)*.

[27] Ed Grochowski, David Ayers, and Vivek Tiwari. 2002. Microarchitectural simulation and control of di/dt-induced power supply voltage variation. In *HPCA*.

[28] Zacharias Hadjilambrou, Shidhartha Das, Paul N Whatmough, David Bull, and Yiannakis Sazeides. 2019. GeST: An automatic framework for generating CPU stress-tests. In *ISPASS*.

[29] Jawad Haj-Yihia, Ahmad Yasin, Yosi Ben Asher, and Avi Mendelson. 2016. Fine-grain power breakdown of modern out-of-order cores and its implications on skylake-based systems. *TACO* (2016).

[30] Charles R. Harris, K. Jarrod Millman, St'efan J. van der Walt, Ralf Gommers, Pauli Virtanen, David Cournapeau, et al. 2020. Array programming with NumPy. *Nature* (2020).

[31] Arthur E Hoerl and Robert W Kennard. 1970. Ridge regression: applications to nonorthogonal problems. *Technometrics* (1970).

[32] Chang-Hong Hsu, Yunqi Zhang, Michael A Laurenzano, David Meisner, Thomas Wenisch, Jason Mars, Lingjia Tang, and Ronald G Dreslinski. 2015. Adrenaline: Pinpointing and reining in tail queries with quick voltage boosting. In *HPCA*.

[33] Wei Huang, Charles Lefurgy, William Kuk, Alper Buyuktosunoglu, Michael Floyd, Karthick Rajamani, Malcolm Allen-Ware, and Bishop Brock. 2012. Accurate fine-grained processor power proxies. In *MICRO*.

[34] Canturk Isci and Margaret Martonosi. 2003. Runtime power monitoring in high-end processors: Methodology and empirical data. In *MICRO*.

[35] Hans Jacobson, Alper Buyuktosunoglu, Pradip Bose, Emrah Acar, and Richard Eickemeyer. 2011. Abstraction and microarchitecture scaling in early-stage power modeling. In *HPCA*.

[36] R. Joseph and M. Martonosi. 2001. Run-time power estimation in high performance microprocessors. In *ISLPED*.

[37] Vijay Kiran Kalyanam, Eric Mahurin, Keith Bowman, and Jacob Abraham. 2020. A Proactive Voltage-Droop-Mitigation System in a 7nm Hexagon™ Processor. In *VLSI*.

[38] Vijay Kiran Kalyanam, Peter G Sassone, and Jacob A Abraham. 2017. Power prediction of embedded scalar and vector processor: Challenges and solutions. In *ISQED*.

[39] Harshad Kasture, Davide B Bartolini, Nathan Beckmann, and Daniel Sanchez. 2015. Rubik: Fast analytical power management for latency-critical systems. In *MICRO*.

[40] Donggyu Kim, Jerry Zhao, Jonathan Bachrach, and Krste Asanović. 2019. Simmani: Runtime Power Modeling for Arbitrary RTL with Automatic Signal Selection. In *MICRO*.

[41] Thomas N. Kipf and Max Welling. 2017. Semi-Supervised Classification with Graph Convolutional Networks. In *ICLR*.

[42] Ajay Krishna Ananda Kumar and Andreas Gerstlauer. 2019. Learning-Based CPU Power Modeling. In *MLCAD*.

[43] Benjamin C Lee and David M Brooks. 2006. Accurate and efficient regression modeling for microarchitectural performance and power prediction. *ACM SIGOPS operating systems review* (2006).

[44] Dongwook Lee, Lizy K John, and Andreas Gerstlauer. 2015. Dynamic power and performance back-annotation for fast and accurate functional hardware simulation. In *DATE*.

[45] Wooseok Lee, Youngchun Kim, Jee Ho Ryoo, Dam Sunwoo, Andreas Gerstlauer, and Lizy K John. 2015. PowerTrain: A learning-based calibration of McPAT power models. In *ISLPED*.

[46] Joseph Lee Rodgers and W Alan Nicewander. 1988. Thirteen ways to look at the correlation coefficient. *The American Statistician* (1988).

[47] Charles R Lefurgy, Alan J Drake, Michael S Floyd, Malcolm S Allen-Ware, Bishop Brock, Jose A Tierno, and John B Carter. 2011. Active management of timing guardband to save energy in POWER7. In *MICRO*.

[48] 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*.

[49] Hugh Mair, Ericbill Wang, Alice Wang, Ping Kao, Yuwen Tsai, Sumanth Gururajarao, Rolf Lagerquist, Jin Son, Gordon Gammie, Gordon Lin, et al. 2017. 3.4 A 10nm FinFET 2.8GHz tri-gear deca-core CPU complex with optimized power-delivery network for mobile SoC performance. In *ISSCC*.

[50] Nico JD Nagelkerke et al. 1991. A note on a general definition of the coefficient of determination. *Biometrika* (1991).

[51] Mohamad Najem, Pascal Benoit, Mohamad El Ahmad, Gilles Sassatelli, and Lionel Torres. 2017. A design-time method for building cost-effective run-time power monitoring. *IEEE TCAD* (2017).

[52] Fabian Oboril, Jos Ewert, and Mehdi B Tahoori. 2015. High-resolution online power monitoring for modern microprocessors. In *DATE*.

[53] Daniele Jahier Pagliari, Valentino Peluso, Yukai Chen, Andrea Calimera, Enrico Macii, and Massimo Poncino. 2018. All-digital embedded meters for on-line power estimation. In *DATE*.

[54] Neal Parikh and Stephen Boyd. 2014. Proximal algorithms. *Foundations and Trends in optimization* (2014).

[55] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. 2019. PyTorch: An Imperative Style, High-Performance Deep Learning Library. *NeurIPs* (2019).

[56] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, et al. 2011. Scikit-learn: Machine Learning in Python. *JMLR* (2011).

[57] Andrea Pellegrini, Nigel Stephens, Magnus Bruce, Yasuo Ishii, Joseph Pusdesris, Abhishek Raja, Chris Abernathy, Jinson Koppanalil, Tushar Ringe, Ashok Tummala, et al. 2020. The Arm Neoverse N1 Platform: Building Blocks for the

Next-Gen Cloud-to-Edge Infrastructure SoC. *IEEE Micro* (2020).
[58] Mihai Pricopi, Thannirmalai Somu Muthukaruppan, Vanchinathan Venkataramani, Tulika Mitra, and Sanjay Vishin. 2013. Power-performance modeling on asymmetric multi-cores. In *CASES*.
[59] Vijay Janapa Reddi, Meeta S Gupta, Glenn Holloway, Gu-Yeon Wei, Michael D Smith, and David Brooks. 2009. Voltage emergency prediction: Using signatures to reduce operating margins. In *HPCA*.
[60] Vijay Janapa Reddi, Svilen Kanev, Wonyoung Kim, Simone Campanoni, Michael D Smith, Gu-Yeon Wei, and David Brooks. 2010. Voltage noise in production processors. *IEEE micro* (2010).
[61] Santhosh Kumar Rethinagiri, Oscar Palomar, Rabie Ben Atitallah, Smail Niar, Osman Unsal, and Adrian Cristal Kestelman. 2014. System-level power estimation tool for embedded processor based platforms. In *RAPIDO*.
[62] Rance Rodrigues, Arunachalam Annamalai, Israel Koren, and Sandip Kundu. 2013. A study on the use of performance counters to estimate power in microprocessors. *IEEE Transactions on Circuits and Systems II: Express Briefs (TCAS-II)* (2013).
[63] Mark Sagi, Nguyen Anh Vu Doan, Martin Rapp, Thomas Wild, Jörg Henkel, and Andreas Herkersdorf. 2020. A Lightweight Nonlinear Methodology to Accurately Model Multicore Processor Power. *IEEE TCAD* (2020).
[64] Yakun Sophia Shao, Brandon Reagen, Gu-Yeon Wei, and David Brooks. 2014. Aladdin: A pre-rtl, power-performance accelerator simulator enabling large design space exploration of customized architectures. In *ISCA*.
[65] Karan Singh, Major Bhadauria, and Sally A McKee. 2009. Real time power estimation and thread scheduling via performance counters. *ACM SIGARCH Computer Architecture News* (2009).
[66] Dam Sunwoo, Gene Y Wu, Nikhil A Patil, and Derek Chiou. 2010. PrEsto: An FPGA-accelerated power estimation methodology for complex systems. In *FPL*.
[67] Robert Tibshirani. 1996. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society: Series B (Methodological)* (1996).
[68] Matthew J Walker, Stephan Diestelhorst, Andreas Hansson, Anup K Das, Sheng Yang, Bashir M Al-Hashimi, and Geoff V Merrett. 2016. Accurate and Stable Run-Time Power Modeling for Mobile and Embedded CPUs. *IEEE TCAD* (2016).
[69] T Webel, PM Lobo, T Strach, PB Parashurama, S Purushotham, R Bertran, and A Buyuktosunoglu. 2020. Proactive power management in IBM z15. *IBM Journal of Research and Development* (2020).
[70] Fei Wen, Lei Chu, Peilin Liu, and Robert C Qiu. 2018. A survey on nonconvex regularization-based sparse and low-rank recovery in signal processing, statistics, and machine learning. *IEEE Access* (2018).
[71] Stephen J Wright. 2015. Coordinate descent algorithms. *Mathematical Programming* (2015).
[72] Qing Wu, Qinru Qiu, Massoud Pedram, and Chih-Shun Ding. 1998. Cycle-accurate macro-models for RT-level power analysis. *VLSI* (1998).
[73] Sam Likun Xi, Hans Jacobson, Pradip Bose, Gu-Yeon Wei, and David Brooks. 2015. Quantifying sources of error in McPAT and potential impacts on architectural studies. In *HPCA*.
[74] Zhang Xuegong. 2000. Introduction to statistical learning theory and support vector machines. *Acta Automatica Sinica* (2000).
[75] Jianlei Yang, Liwei Ma, Kang Zhao, Yici Cai, and Tin-Fook Ngai. 2015. Early stage real-time SoC power estimation using RTL instrumentation. In *ASPDAC*.
[76] Wu Ye, Narayanan Vijaykrishnan, Mahmut Kandemir, and Mary Jane Irwin. 2000. The design and use of SimplePower: A cycle-accurate energy estimation tool. In *DAC*.
[77] Cun-Hui Zhang. 2010. Nearly unbiased variable selection under minimax concave penalty. *The Annals of statistics* (2010).
[78] Yanqing Zhang, Haoxing Ren, and Brucek Khailany. 2020. GRANNITE: Graph Neural Network Inference for Transferable Power Estimation. In *DAC*.
[79] Yuan Zhou, Haoxing Ren, Yanqing Zhang, Ben Keller, Brucek Khailany, and Zhiru Zhang. 2019. PRIMAL: Power Inference using Machine Learning. In *DAC*.
[80] Davide Zoni, Luca Cremona, Alessandro Cilardo, Mirko Gagliardi, and William Fornaciari. 2018. PowerTap: All-digital power meter modeling for run-time power monitoring. *Elsevier Microprocessors and Microsystems (MICPRO)* (2018).
[81] Davide Zoni, Luca Cremona, and William Fornaciari. 2018. Powerprobe: Run-time power modeling through automatic RTL instrumentation. In *DATE*.
[82] Yazhou Zu, Charles R Lefurgy, Jingwen Leng, Matthew Halpern, Michael S Floyd, and Vijay Janapa Reddi. 2015. Adaptive guardband scheduling to improve system-level efficiency of the POWER7+. In *MICRO*.
[83] Yazhou Zu, Daniel Richins, Charles Lefurgy, and Vijay Reddi. 2019. Fine-tuning the active timing margin (ATM) control loop for maximizing multi-core efficiency on an IBM POWER server. In *HPCA*.