An Architecture-Level CPU Modeling Framework for Power and Other Design Qualities

Qijun Zhang, Mengming Li, Andrea Mondelli, Zhiyao Xie, *Member, IEEE*

***Abstract*—Power efficiency is a critical design objective in modern microprocessor design. To evaluate the impact of architectural-level design decisions, an accurate yet efficient architecture-level power model is desired. However, widely adopted analytical power models like McPAT and Wattch have been criticized for their unreliable accuracy, while machine learning (ML) methods like McPAT-Calib rely on sufficient known designs for training and perform poorly when available designs are limited, which is the case in realistic scenarios.**

In this work, we propose PANDA, an innovative architecture-level solution that combines the advantages of analytical and ML power models. It achieves unprecedented high accuracy on unknown new designs even when there are very limited designs for training. Besides being an excellent average power model, We also extend PANDA to support the time-based power trace prediction, which can enable the analysis of peak power, power fluctuations, and voltage fluctuation. This is highly challenging at the architecture level. Other qualities such as area, performance, and energy accurately can also be supported. In addition to single design quality, PANDA can model the trade-offs among different design qualities such as the trade-off between power and timing by predicting the Pareto-optimal curve. Finally, PANDA can further support power prediction for unknown new technology nodes. Our experiment shows that, for average power prediction, our method can achieve high accuracy with a correlation coefficient R of 0.99 and mean absolute percentage error (MAPE) of 7.91% even when only one configuration is known, outperforming McPAT-Calib which has R of -0.24 and MAPE of 35.96%. For time-based power trace prediction, our method can achieve a low MAPE of 4.34%, outperforming the state-of-the-art method Powertrain which has a MAPE of 53.8%.

I. Introduction

Power efficiency is a key objective in microprocessor architecture design. As chip complexity keeps increasing, designing microprocessors for higher power efficiency requires huge efforts and a long turnaround time. As a result, there is a high demand for fast yet accurate early-stage power modeling methodologies, which are essential for effective early design optimizations. For example, before the register-transfer level (RTL) implementation, chip architects need to efficiently analyze the power efficiency of many different design configurations at the architecture level.

However, traditional power modeling approaches cannot well satisfy these requirements. Conventional VLSI design flow delivers accurate power estimations after going through several design stages, including RTL implementation, logic synthesis, RTL simulation with workloads, and gate-level power simulation using commercial tools [1], [2]. It is prohibitively time-consuming for designers to accurately measure each architectural-level design variant. In terms of faster alternatives, commonly used architectural-level power models such as McPAT [3]–[5] and Wattch [6] have been widely criticized for their unreliable accuracies in several researches [7], [8]. For example, our experiment has measured that, for RISC-V BOOM, even adopting the McPAT scaled towards the ground truth will have an error with MAPE of 15%, and the error can reach 1234% without the scaling. Despite certain advancements in following works, they are typically only created in-house to meet the needs of proprietary designs [7].

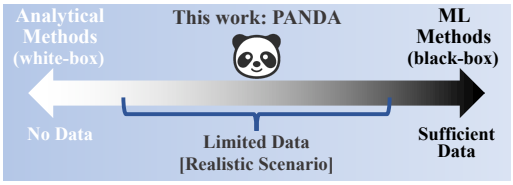


Fig. 1: A summary of architecture-level power modeling methods. Analytical methods adopt inaccurate oversimplified handcrafted models, while the accuracies of ML methods degrade significantly when the number of training data is limited. PANDA unifies both analytical and ML solutions, addressing their long-lasting limitations.

In recent years, several ML methods [11], [13], [14] have been proposed to directly calibrate analytical models like McPAT [3]. These methods mainly use McPAT's output as model input, aiming to generate more accurate estimations, especially when the target design architecture is similar to certain designs in training dataset. However, when applied to unknown new design configurations, these methods have a significant accuracy drop. This problem is particularly serious when training data is limited, which is often the case in practical scenarios. As mentioned above, collecting the label of each new sample, which requires implementation with VLSI flow and workload-based simulations, is a time-consuming process. A recent ML approach [14] proposes using transfer learning to predict unknown new designs. Nevertheless, it still requires a few ground-truth samples in each target configuration domain, which can also be time-consuming to generate. Additionally, certain design space exploration (DSE) works [15], [16] develop their own ML-based power models through iterative training based on labels collected during exploration. Besides still being constrained by training data, they typically cannot incorporate workload-related information, thus failing to predict each workload-specific power.

In this work, we will first present our qualitative analysis of existing analytical [3], [6] and ML-based [11], [13]–[16]

Manuscript received XXXX; revised XXXX; accepted XXXX. Date of publication XXXX; date of current version XXXX. This work is sponsored by the National Natural Science Foundation of China (NSFC) 62304192 and ACCESS – AI Chip Center for Emerging Smart Systems, sponsored by InnoHK funding, Hong Kong SAR. *Corresponding Author: Zhiyao Xie.*

Qijun Zhang, Mengming Li, and Zhiyao Xie are with the Department of Electronic and Computer Engineering at the Hong Kong University of Science and Technology, Hong Kong SAR (email: qzhangcs@connect.ust.hk; mengming.li@connect.ust.hk; eezhiyao@ust.hk).

Works	Power					Other Design Qualities			
	Component Power	Cross-Config	Cross-Domain	Cross-Tech Node	Time-based Waveform	Gate Area	Timing (Slack)	Performance (# Cycles)	Design Quality Trade-off
Gem5 [9] / Sniper [10]								✓	
Gem5 [9] / Sniper [10] + McPAT [3]	✓	✓	✓	✓		✓	✓	✓	
Wattch [6]	✓	✓	✓	✓	✓				
PowerTrain [11]	✓				✓⊛				
TCAD'20 [12]									
McPAT-Calib [13]		✓							
ASP-DAC'23 [14]		✓	✓⋆						
PANDA (Ours)	✓	✓	✓	✓	✓	✓	✓	✓	✓

TABLE I: Comparison of existing microarchitecture-level design quality models. We primarily focus on power modeling methods. An empty cell means the functionality (at column) is not originally proposed by the work (at row). ⋆The work of [14] is not completely cross-domain, and still requires the ground-truth labels of a few samples in the target new domain. ⊛The work of [11] can only generate the time-based power waveform at a very coarse-grained temporal resolution.

Method	#Known Config = 1		#Known Config = 14	
	MAPE(%)	R	MAPE(%)	R
McPAT (Analytical)	1234	0.89	1234	0.89
McPAT-plus (Analytical)	14.81	0.89	14.81	0.89
McPAT-Calib (ML-based)	35.96	-0.24	5.7	0.98

TABLE II: Comparison between two existing methods, McPAT and McPAT-Calib, where McPAT-plus is the McPAT scaled towards ground truth. It shows that the analytical model has low accuracy while the ML-based model relies on sufficient training data.

architecture-level models, as summarized in Fig. 1. The performance of ML-based models is superior when there is sufficient training data that covers the potential testing scenarios. However, limited data availability can lead to misleading ML-based models, necessitating the use of traditional analytical methods. In practice, design teams often face a situation where they have a limited number of implemented architecture configurations available for training data. The demand for diverse and abundant training data poses a significant barrier to the widespread adoption of ML solutions [11], [13]–[16] in practice. Table II quantifies this analysis. It shows that the analytical models have low accuracy and the ML-based model relies on sufficient training data to achieve a high accuracy.

Inspired by these observations, we introduce a novel architectural-level power modeling approach named PANDA. As depicted in Fig. 1, the name PANDA embodies the integration of white-box analytical models and black-box ML models, leveraging the complementary strengths of both approaches, like the panda with both white and black fur. PANDA employs an analytical framework to capture the hierarchy of individual components. For each component, it combines an ML model with a customized analytical function based on essential configuration parameters. This analytical function captures the core behavior of the component, while more intricate patterns are learned by the ML Model. Consequently, PANDA outperforms existing solutions, particularly in scenarios with limited training data. Its minimal data requirements facilitate its potential widespread adoption. Notably, unlike most existing ML techniques [11], [13]–[16] that rely on established analytical models, PANDA stands as an independent solution, unencumbered by dependencies on existing models.

Besides average power, architecture-level time-based power modeling is also essentially useful in modern processor evaluation and optimization. In contrast to average power, time-based power trace provides power information for each time step, allowing detailed power-related analysis. For example, such fine-grained power information enables the analysis of peak power, power fluctuation, and voltage droops, which are the premises of most mitigation and optimization techniques. With modern CPUs hitting the "power wall", the mitigation of higher-than-threshold peak power and drastic power fluctuations is vitally important. However, because of the lack of RTL-level information, i.e. the RTL implementation and the toggling of signals, the architecture-level time-based power trace modeling is highly challenging. Wattch [6] is a widely adopted tool for architecture-level time-based power trace analysis, but the accuracy is unreliable. This is because the model and parameters are outdated compared with modern CPU designs, and some detailed component analyses are missing.

What's more, to comprehensively evaluate a design at the architecture level, evalu ations of other design qualities like performance (i.e., number of cycles to complete a workload) , area, and energy are also desired. For performance, cycle-level simulators like gem5 [9] and Sniper [10] are widely adopted, but it is not sufficiently accurate. For gate area, McPAT [3] is widely-adopted but inaccurate. More importantly, in modern VLSI design flow, trade-offs between power and other design qualities widely exist when using different downstream synthesis options. Therefore, the same architecture design configuration will not necessarily lead to a fixed power value after implementation. But most existing ML-based power models [11], [13], [14] do not consider such trade-offs. To solve these challenges, our solution PANDA also enables accurate evaluations of other design objectives.

Table I shows the comparison of existing architecture-level design quality models and PANDA. It qualitatively compares PANDA with existing works in the modeling of power and other design qualities (e.g., area, timing, and performance). Most of the existing works [3], [6], [9]–[14] only focus on a small number of application scenarios. The only exception, gem5 [9] / Sniper [10] + McPAT [3], targets modeling both the power and other design qualities, but the accuracy is very limited. Also, even gem5/Sniper+McPAT cannot model time-based power traces and the trade-off between different design qualities. In comparison, PANDA is a comprehensive framework that targets multiple design quality evaluation problems.

The key contributions can be summarized below.

- We analyzed the root cause of limited accuracy in both analytical and ML-based power models, then propose an open-sourced architecture-level power modeling solution named PANDA[1], which unifies these two major types of methods. PANDA demonstrates superior performance

[1]It has been open-sourced at https://github.com/hkust-zhiyao/PANDA

Component i	Configuration parameters C_i of each component	Event parameters E_i of each component	CPU part
BP	FetchWidth, BranchCount	BTBLookups, condPredicted, condIncorrect, commit.branches	Frontend
IFU	FetchWidth, DecodeWidth FetchBufferEntry, ICacheFetchBytes	fetch.insts, fetch.branches, fetch.cycles, numRefs, numStoreInsts, numInsts, decode.runCycles, decode.blockedCycles, decode.decodedInsts, numBranches, intInstQueueReads, intInstQueueWrites, intInstQueueWakeupAccesses, fpInstQueueReads, fpInstQueueWrites, fpInstQueueWakeupAccesses	
I-TLB	ICacheTLBEntry	itb.accesses, itb.misses	
I-Cache	ICacheWay, ICacheFetchBytes	icache.overallAccesses, icache.overallMisses, icache.ReadReq.mshrHits, icache.ReadReq.mshrMisses, icache.tagAccesses	
RNU	DecodeWidth	intLookups, renamedOperands, fpLookups, renamedInsts, runCycles, blockCycles, committedMaps	Execution
ROB	DecodeWidth, RobEntry	rob.reads, rob.writes	
ISU	DecodeWidth, MemIssueWidth, FpIssueWidth, IntIssueWidth	IssuedMemRead, IssuedMemWrite, IssuedFloatMemRead, IssuedFloatMemWrite, IssuedIntAlu, IssuedIntMult, IssuedIntDiv, IssuedFloatMult, IssuedFloatDiv	
Regfile	DecodeWidth, IntPhyRegister, FpPhyRegister	intRegfileReads, fpRegfileReads, intRegfileWrites, fpRegfileWrites, functionCalls	
FU Pool	MemIssueWidth, FpIssueWidth, IntIssueWidth	intAluAccesses, fpAluAccesses	
LSU	LDQEntry, STQEntry, MemIssueWidth	MemRead, InstPrefetch, MemWrite	Mem Access
D-TLB	DCacheTLBEntry	dtb.accesses, dtb.misses	
D-Cache	DCacheWay, DCacheTLBEntry, DCacheMSHR, MemIssueWidth	dcache.ReadReq.accesses, dcache.WriteReq.accesses, dcache.ReadReq.misses, dcache.WriteReq.misses, dcache.overallMisses, dcache.MshrHits, dcache.MshrMisses, dcache.tagAccesses	
Other Logic	All	All	Other Logic

TABLE III: Our identified architecture-level design configuration parameters C_i and event parameters E_i of each i^{th} component.

compared to state-of-the-art baselines, achieving a reduction in absolute error by 5% to 30%. The improvement is especially obvious when the training data is limited.

- We also enable PANDA for fine-grained time-based power traces, which can enable the analysis of peak power, power fluctuations, and voltage fluctuation (e.g., Ldi/dt). This is highly challenging at the architecture level, considering the lack of RTL implementation or RTL simulation. Most existing works only support estimating the average power for each target workload.
- Besides power, PANDA also models other design objectives including design performance, area, and energy at the architecture level. It can also model the trade-offs among different design qualities such as the trade-off between power and timing by predicting the Pareto-optimal curve. Finally, PANDA further supports the power prediction targeting unknown new technology nodes.

II. Formulation of Existing Work

In this section, we provide a general formulation of all existing power models. Fig. 4 represents analytical models [3], [6], ML-based models [11], [13], [14], and PANDA, in a consistent framework. We observe that these power modeling methods can be unified into a general formulation. Therefore, we begin by presenting our formulation of two existing approaches, followed by introducing our novel method in the next section.

Architecture-level power modeling takes architecture-level configuration parameters for each component and event parameters for each component as the model input to estimate power. These configuration parameters, denoted as set C, are the representative parameters that can describe the scale of each component in the CPU. The event parameters, denoted as set E, are related to the workload executed on each component. Assuming there are N components modeled in our targeted CPU design, for the i^{th} component, we denote related configuration parameters as C_i, with $C = \{C_i \,|\, i \in [1, N]\}$, and related event parameters as E_i, with $E = \{E_i \,|\, i \in [1, N]\}$.

Formulation of ML works. Existing ML solutions [11], [13], [14] build ML models targeting total power, based on all available design configuration parameters C and event parameters E.[2] $\boldsymbol{F_{ml}}$ denotes ML methods. It can be formulated below, with P_{ml} denoting power prediction.

$$P_{\text{ml}} = \boldsymbol{F_{ml}}\ (\{C, E\})$$

It can be rewritten as an equivalent general form by explicitly indicating configuration parameters of all components:

$$P_{\text{ml}} = \boldsymbol{F_{ml}}\ (\{C_i \,|\, i \in [1, N]\},\ \{E_i \,|\, i \in [1, N]\}) \tag{1}$$

It means existing ML methods adopt the available configuration parameters and event parameters information from all components to evaluate the total power of the whole design.

Formulation of analytical works. Different from ML methods, analytical methods like McPAT [3] explicitly design separate analytical models for each component, whose estimated power is denoted as P^i_{ana}, according to designers' background knowledge. We formulate such analytical methods for each component i as below,

$$P^i_{\text{ana}} = \boldsymbol{F^i_{agg}}\ (E_i,\ \boldsymbol{F^i_{res}}(C_i)) \tag{2}$$

where an analytical 'resource function' $\boldsymbol{F^i_{res}}(C_i)$ first calculates a resource value that reflects the resource consumption based on configuration parameters. Then 'aggregation function' $\boldsymbol{F^i_{agg}}$ combines both resource values $\boldsymbol{F^i_{res}}(C_i)$ and event parameters E_i to calculate component power.

We illustrate the aforementioned analytical methods with the I-Cache component in CPU frontend as an example. The configuration parameters C_i of I-Cache include the number of ways of the N-set associated cache (i.e., *ICacheWay*) and the unit of line capacity that I-Cache supports (i.e., *ICacheFetchBytes*). Analytical models like McPAT compute the power based on the number of hits and misses. We can formulate its resource function $\boldsymbol{F^i_{res}}$ as estimating the energy per hit and miss based on the I-Cache configuration parameters.

$$\boldsymbol{F^i_{res}}\ (\textit{ICacheWay},\ \textit{ICacheFetchBytes}) = \textit{Energy per hit/miss} \tag{3}$$

Then aggregation function $\boldsymbol{F^i_{agg}}$ combines the resources and corresponding event parameters, including the number of hits and misses. Then the actual implementation of Equation (2) for I-Cache component can be expressed as below.

$$\begin{aligned} P^i_{\text{ana}} &= \boldsymbol{F^i_{agg}}\ (\textit{\#Hit, \#Miss, Energy per hit/miss}) \\ &= \frac{\textit{\#Hit} * \textit{Energy per hit} + \textit{\#Miss} * \textit{Energy per miss}}{\textit{Total benchmark execution time}} \end{aligned}$$

Based on predicted component power, the total power is simply the summation of all components $P_{\text{ana}} = \sum_{i \in [1,N]} P^i_{\text{ana}}$.

[2] For simplicity, we do not include the McPAT output as a potential input feature in the formulation of ML works.

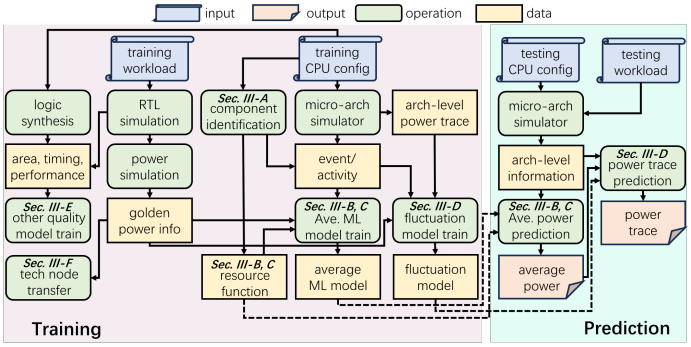


Fig. 2: The overview of PANDA. The workflow of PANDA has two stages including training and prediction. Our contributions and corresponding sections have been shown.

III. Methodology

Fig. 2 shows the overview of PANDA framework, which supports the power modeling including average power and time-based power trace, other quality prediction, and cross-technology node transfer. The framework includes the training and prediction stages. In the training stage, we collect features from the input configuration and architecture-level simulators and generate labels with standard VLSI flow with EDA tools. In the prediction stage, with the feature collected from the architecture-level simulator, we can utilize the models obtained in the training stage for estimation. As shown in this figure, Sec. III-A introduces component identification. Then Sec. III-B and III-C discuss the average power modeling. Based on average power modeling, Sec. III-D describes the time-based power trace prediction. Finally, the other quality prediction and cross-technology node transfer will be discussed in Sec. III-E and III-F respectively.

A. CPU Components and Parameters Identification

For the target out-of-order (OoO) CPU[3] in our study, we identify essential CPU components that can be independently modeled for power evaluation. Fig. 3 shows overall CPU architecture including these components, which can be categorized into three parts: frontend, execution, and memory access. Each part consists of several key components, as introduced below.

- The CPU frontend includes branch predictor (BP), instruction fetch unit (IFU), instruction translation-lookaside buffer (I-TLB), L1 instruction cache (I-Cache).
- The CPU execution part includes the rename unit (RNU), reorder buffer (ROB), issue unit (ISU), register file (Reg-file), and functional unit pool (FU Pool), including ALUs, floating-point units, and other functional units.
- The CPU memory-access part includes data translation-lookaside buffer (D-TLB), the data cache (D-Cache), and the remaining logic in the load/store unit (LSU).

Any CPU design logic not covered by the components mentioned above is referred to as *Other Logic*.

Table III presents the widely available architecture-level configuration and event parameters for each component.

[3]PANDA experiments on the RISC-V OoO CPU core BOOM [17]. It can be extended to other CPU designs with minor modifications.

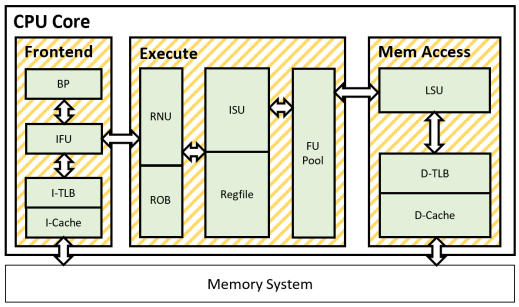


Fig. 3: The architecture of our target Out-of-Order RISC-V CPU core. The solid-filled blocks indicate key individual components modeled by PANDA. These components correspond to the Table III. The slash-filled block refers to the Other Logic indicated in Table III.

B. Formulation of Average Power Modeling in PANDA

Formulation of PANDA. In contrast with ML methods in Equation (1) and analytical methods in Equation (2), we combine the advantages of both methods in this work. The general expression for each component i is shown below.

$$P^i_{\text{PANDA}} = \boldsymbol{F}^i_{\boldsymbol{agg}}\left(\boldsymbol{F}^i_{\boldsymbol{ml}}(C_i, E_i),\ \boldsymbol{F}^i_{\boldsymbol{res}}(C_i)\right)$$

Similar to the notations used in Equation (1)(2), here the $\boldsymbol{F}^i_{\boldsymbol{agg}}$ and $\boldsymbol{F}^i_{\boldsymbol{res}}$ denote analytical functions, and $\boldsymbol{F}^i_{\boldsymbol{ml}}$ denotes an ML-based function. The general formulation combines the ML model in Equation (1) and the analytical model in Equation (2). Notably, it is a formulation rather than a specific implementation for power modeling methods, so the $\boldsymbol{F}^i_{\boldsymbol{ml}}$ and $\boldsymbol{F}^i_{\boldsymbol{res}}$ here are different from the $\boldsymbol{F}^i_{\boldsymbol{ml}}$ described for ML-based power model and the $\boldsymbol{F}^i_{\boldsymbol{res}}$ for analytical power model above. Now we start to introduce each part and explain its advantages.

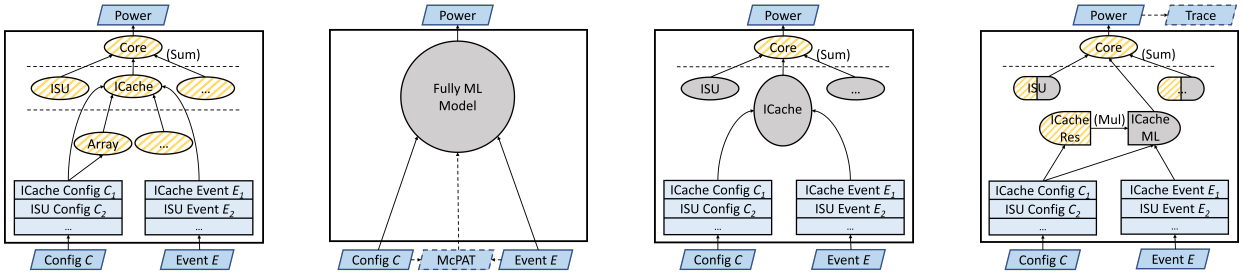
First, we design an analytical resource function $\boldsymbol{F}^i_{\boldsymbol{res}}(C_i)$ according to background knowledge of how the configuration parameters C_i will affect the power of this component. Compared with the similar function in Equation (2), we capture the simpler yet primary pattern in this function, and leave complex patterns to be learned by our ML model.

Using the same I-Cache component example, for a typical N set-associative I-Cache, each cache access requires accessing both tag and data array in all cache ways simultaneously for lower latency. It causes the power consumption to be roughly proportional to the number of cache ways (i.e., *ICacheWay*). Regarding the *ICacheFetchBytes*, it decides the size of the cache line of the I-Cache, so the power of accessing a cache line in a way will scale proportionally with it. Considering both factors, our resource function is as below.

$$\boldsymbol{F}^i_{\boldsymbol{res}}(C_i) = \mathit{ICacheWay} * \mathit{ICacheFetchBytes} \tag{4}$$

The resource function is a function of configuration parameters rather than a constant because the configuration parameters are different for different configurations.

Second, we propose the ML model $\boldsymbol{F}^i_{\boldsymbol{ml}}(C_i, E_i)$ based on both configuration parameters and event parameters for each component i. It learns all the detailed correlations beyond the simple correlation in resource function. Finally, the estimations of ML model $\boldsymbol{F}^i_{\boldsymbol{ml}}$ and resource function $\boldsymbol{F}^i_{\boldsymbol{res}}$ are multiplied to generate the final power estimation. The finalized PANDA formulation is shown below.



(a) Analytical power model (b) ML-based power model (c) Component-level ML model (d) PANDA

Fig. 4: Illustration of different power modeling methods. Slash-filled blocks mean analytical parts and solid-filled blocks mean ML parts. (a) The analytical method [3], [6]. (b) The ML method [11], [13]–[16]. (c) Component-level power model, with ML-based power models for each component. (d) PANDA, slash-filled blocks mean analytical resource functions, while solid-filled blocks mean ML models. It can also be extended for time-based power trace prediction.

Component i	Resource function $\boldsymbol{F}^i_{\boldsymbol{res}}(C_i)$
ICache	$\boldsymbol{F}_{\boldsymbol{res}}$ = ICacheWay * ICacheFetchBytes
BP	$\boldsymbol{F}_{\boldsymbol{res}}$ = FetchWidth
ISU	$\boldsymbol{F}_{\boldsymbol{res}}$ = $f_{\text{ReserveStationNum}}$(DecodeWidth)
IFU	$\boldsymbol{F}_{\boldsymbol{res}}$ = DecodeWidth
Regfile	$\boldsymbol{F}_{\boldsymbol{res}}$ = IntPhysRegister + FpPhysRegister
I-TLB	$\boldsymbol{F}_{\boldsymbol{res}}$ = ICacheTLBEntry + bias
DCache	$\boldsymbol{F}_{\boldsymbol{res}}$ = DCacheWay * MemIssueWidth
RNU	$\boldsymbol{F}_{\boldsymbol{res}}$ = DecodeWidth
LSU	$\boldsymbol{F}_{\boldsymbol{res}}$ = LDQEntry + STQEntry
ROB	$\boldsymbol{F}_{\boldsymbol{res}}$ = RobEntry
D-TLB	$\boldsymbol{F}_{\boldsymbol{res}}$ = DCacheTLBEntry + bias
FU Pool	$\boldsymbol{F}_{\boldsymbol{res}}$ = 1
Other Logic	$\boldsymbol{F}_{\boldsymbol{res}}$ = DecodeWidth + bias

TABLE IV: PANDA's resource function $\boldsymbol{F}^i_{\boldsymbol{res}}$ of each major component i in the target out-of-order CPU core.

$$\begin{aligned} P^i_{\text{PANDA}} &= \boldsymbol{F}^i_{\boldsymbol{agg}}\ (\boldsymbol{F}^i_{\boldsymbol{ml}}(C_i, E_i),\ \boldsymbol{F}^i_{\boldsymbol{res}}(C_i)) \\ &= \boldsymbol{F}^i_{\boldsymbol{ml}}(C_i, E_i) * \boldsymbol{F}^i_{\boldsymbol{res}}\ (C_i) \end{aligned} \tag{5}$$

As Equation (5) shows, PANDA adopts a simple multiplication to aggregate the ML model $\boldsymbol{F}^i_{\boldsymbol{ml}}$ and analytical resource function $\boldsymbol{F}^i_{\boldsymbol{res}}$. But there may not be a definitive answer. The detailed discussion can be found in [18].

Substituting the resource function $\boldsymbol{F}^i_{\boldsymbol{res}}\ (C_i)$ in Equation (5) with Equation (4), the power of the I-Cache example is:

$$P^i_{\text{PANDA}} = \boldsymbol{F}^i_{\boldsymbol{ml}}(C_i, E_i) * \mathit{ICacheWay} * \mathit{ICacheFetchBytes} \tag{6}$$

C. Resource Functions and ML Model for Average Power

Table IV shows our proposed key resource functions $\boldsymbol{F}^i_{\boldsymbol{res}}$ for all major components in the target out-of-order CPU core. We derive these functions based on our architecture-level analysis that the power consumption of each component correlates with a function of configuration parameters, which is the rationale of resource function. The detailed analysis is described below. It is also validated in Sec. V-A. We also observe that the resource function can lower the difficulty of the ML model by changing the data distribution, which will be discussed in Sec. V-B.

Similar to the I-Cache example, the power consumption of an N-way set-associative L1 data cache (D-Cache) exhibits a rough proportionality to the number of cache ways (i.e., *DCacheWay*). Additionally, modern CPUs enhance throughput by concurrently servicing multiple read requests [17]. So the power typically correlates with the number of memory-access instructions issued per cycle (i.e., *MemIssueWidth*). Consequently, we derive $\boldsymbol{F}^i_{\boldsymbol{res}}$ = *DCacheWay* * *MemIssueWidth*.

❶ **Frontend.** The Frontend of the Out-of-Order CPU encompasses four primary components: BP, IFU, I-TLB, and I-Cache. In the previous section, we have already discussed the I-Cache component. Here we introduce our proposed resource functions of some of the remaining components: (1) For the branch predictor (BP), normally the size scales proportionally with the number of instructions fetched each time (i.e., *FetchWidth*) at the frontend. Therefore we propose $\boldsymbol{F}^i_{\boldsymbol{res}}$ = *FetchWidth* for the BP. (2) For the I-TLB, its power is mainly affected by the number of TLBEntry (i.e., *ICacheTLBEntry*), but there is also a part that remains unchanged when increasing the number of TLBEntry. Therefore, we set $\boldsymbol{F}^i_{\boldsymbol{res}}$ = *ICacheTLBEntry* + bias for I-TLB, with the bias denoting a constant power value. The bias can be estimated by fitting this linear function to the training data.

❷ **Execution.** The Execution part of the Out-of-Order CPU encompasses five main components: RNU, ROB, ISU, Regfile, and FU Pool. Here we explain the design of some representative components: (1) The renaming unit (RNU) typically has a renaming width equal to the *DecodeWidth*. Consequently, we propose the resource function $\boldsymbol{F}^i_{\boldsymbol{res}}$ = *DecodeWidth* for the RNU. (2) The power consumption of the issue unit (ISU) is impacted by the reserve stations. The number of reserve station entries typically depends on the *DecodeWidth*. We define the resource function $\boldsymbol{F}^i_{\boldsymbol{res}}$ = $f_{\text{ReserveStationNum}}$(*DecodeWidth*), where $f_{\text{ReserveStationNum}}$ maps *DecodeWidth* to the number of reserve stations. (3) The FU Pool is a complex component comprising various function units. To handle this complexity, we assign the resource function $\boldsymbol{F}^i_{\boldsymbol{res}}$ = 1 for the FU Pool, deferring the specifics to the ML function.

❸ **Memory Access.** The Memory Access component of the Out-of-Order CPU comprises three main components: LSU, D-TLB, and D-Cache (which has already been discussed). The D-TLB is similar to I-TLB, so we only discuss LSU here: (1) The power consumption of the load store unit (LSU) is closely associated with the total number of entries in these queues, represented as *LDQEntry* + *STQEntry*. Hence, we propose the resource function $\boldsymbol{F}^i_{\boldsymbol{res}}$ = *LDQEntry* + *STQEntry* for LSU.

❹ **Other Logic.** The 'Other Logic' represents the most intricate portion of the CPU. Estimating the power consumption of the 'Other Logic' may initially appear challenging. However, we discovered that the indicator of *DecodeWidth* proves to be valuable in this regard, because *DecodeWidth* serves as a general representation of the pipeline width for the entire CPU design. Consequently, we establish the resource

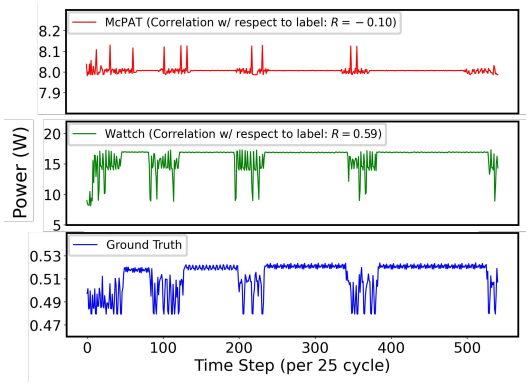


Fig. 5: The comparison between McPAT and Wattch of dev-acc.

function as $\boldsymbol{F^i_{res}}$ = *DecodeWidth* + bias.

However, we would like to mention that this resource function has some limitations especially when the type of workload is special. Our analysis targets the general case across workloads. When the workload shows a special pattern, for example, a workload with a stride access pattern that only accesses the first way of DCache, power will not correlate with *DCacheWay*, so the resource function does not work. Moreover, when applying on a new architecture, the resource function should be redesigned by architects based on the analysis of the target design. We emphasize that the cost of designing a resource function is far lower than that of designing an analytical power model that captures all details in the architecture. This is because the resource function only needs to capture the first-order power characteristics. In this case, our method can still significantly reduce the power modeling efforts and also provide high accuracy.

As for the ML model $\boldsymbol{F^i_{ml}}$ of each component, we all adopt Gradient Boosting Trees [19] like XGBoost [20], a widely-adopted ML algorithm for tabular data type, to build regressors. The Gradient Boosting Trees Regression is an ensemble-learning-based ML algorithm. It builds a regression model based on an ensemble of weak decision tree models. The objective is to investigate the correlation between multiple variables and determine the influential factors that affect the dependent variable. The model is constructed incrementally, enabling the optimization of any differentiable loss function. The hyper-parameter "max depth" determines the complexity of decision trees, and the parameter "num of estimators" determines the number of decision trees utilized.

D. From Average Power to Time-based Power Prediction

To further predict the time-based power trace at the architecture level, we propose a temporal-information-aware method by integrating the average power prediction discussed in previous sections and a new fluctuation model. Our fluctuation model is based on Wattch [6], which is a classical cycle-level power model. In comparison to McPAT, which is built for the average power model and unsuitable for fine-grained time-based power prediction, Wattch uses cycle-level activities of each component to model the time-based cycle-level power trace, thanks to the tight integration with the performance simulator. The cycle-level activities include the accessed units and the number of ports accessed in each cycle. Fig. 5 visualizes the comparison between McPAT and Wattch of dev-acc. It shows that the time-based prediction of Wattch is a better feature than McPAT. Based on these cycle-level activities, Wattch can estimate the per-cycle power for each component. However, because of Wattch's outdated CPU architecture and technology-related parameters, the accuracy of Wattch itself when applied directly to modern CPUs is unreliable when compared with ground-truth labels. Here the ground-truth labels are the power trace generated by the time-based VLSI power simulation flow using EDA tools.

Targeting time-based power prediction, based on predicted average power $P_{average}$, the fluctuation model predicts the power fluctuation $Fluct$ to decide the power within each time step P, i.e., consecutive T cycles, where T is a user-defined parameter deciding the temporal resolution of such time-based power prediction. Our method can be formulated below,

$$P = P_{average} * (1 + Fluct) \tag{7}$$

The fluctuation model is an ML model that captures the "temporal pattern", which reflects a correlation between the fluctuation of the current time step and Wattch-estimated power of the consecutive neighboring time steps. It adopts three types of features for each target time step: (1) The configuration parameters of the BOOM CPU in Table III, including 14 features. They indicate the resources of the target CPU. (2) The activity information of the current time step from Wattch, altogether 19 features. The activity information includes the number of accesses for each component, including renaming unit access, branch predictor access, ROB access, load-store queue access, regfile access, I-Cache access, D-Cache access, ALU access, result bus access, wakeup logic access, and other related logic access. (3) The power value predicted by Wattch for multiple consecutive neighboring time steps. We adopt L time steps before and after the current time step, where L is a hyper-parameter, altogether there are $2L+1$ features, corresponding to $2L+1$ steps of Wattch-predicted power. So there are $14 + 19 + 2L + 1$ features.

To capture the fluctuation, when training the fluctuation model, we propose to obtain the label by normalizing the power label per time step l_{truth} with the average power $l_{average}$ of the whole workload. The l_{truth} is collected by time-based VLSI power simulation flow using EDA tools, and the $l_{average}$ can be calculated from the l_{truth}. The training label of the fluctuation model l_{fluct} is listed below,

$$l_{fluct} = (l_{truth} - l_{average})/l_{average} \tag{8}$$

In this way, the ML model learns power fluctuation directly, without concerning absolute average power value.

For the prediction stage, the output of the fluctuation model trained with Equation 8 will be a normalized value. We will combine the output of the fluctuation model p_{fluct} with the average power $p_{average}$ predicted by the aforementioned average prediction method to get the final result,

$$p_{out} = p_{average} * (1 + p_{fluct}) \tag{9}$$

Because the PANDA's prediction for average power $p_{average}$ is rather accurate, this fluctuation-oriented power model provides reasonably accurate time-based power p_{out} for each time step.

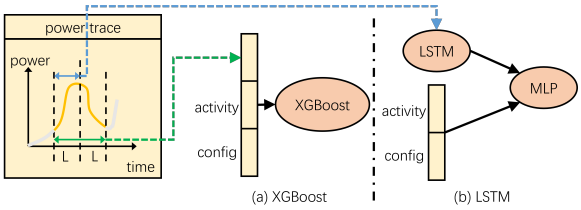


Fig. 6: The illustration of our two ML models. (a) Use XGBoost as ML model, where power trace is treated as tabular data. The power traces of the $2L+1$ time steps are used. (b) Use LSTM as ML model, where power trace is traded as sequential data. The power traces of the L time steps before the current time steps are used.

Then we introduce the specific ML models we have explored for such fluctuation prediction. To capture the correlation between features and the fluctuation per time step, we explore two different ML models. A power trace can be either regarded as tabular data (e.g., a vector) or sequential data (e.g., a sequence of the power values). In the first approach, we treat the power trace as tabular data. XGBoost [20] is one of the most widely adopted models for learning tabular data, which is shown in Fig. 6(a). In the second approach, we trade the power trace as sequential data. LSTM (Long Short-Term Memory) [21] is a representative model for learning sequential data. In this case, we use LSTM to capture the temporal information, as shown in Fig. 6(b). Here we only use the Wattch-generated power values of the L time steps before the current time step. The evaluation of both LSTM and XGBoost is shown in Table VII. We found that training the LSTM for this problem usually leads to overfitting, especially with a small number of training data. The detailed results of the two models will be discussed in Section IV-D.

Finally, we try to qualitatively analyze how the ML solution significantly improves the inaccuracies of Wattch. In general, it is because the ML solution can effectively capture patterns in each target workload program. Fig. 7 shows an example. In this example, for the 'rsort' workload (i.e. radix sort), we observe several power drops in the ground-truth power trace, but it is wrongly evaluated as power rises in the Wattch-generated power trace. The 'rsort' conducts the bucket mapping and collecting stages iteratively, where the mapping is relatively compute-centric while the collecting is relatively memory-centric. The power of the compute-centric stage mainly depends on the power of the pipeline, and the power of the memory-centric stage mainly depends on the power of the cache. In the Wattch, the ratio of the power consumption of the pipeline over the cache is wrongly higher. So in this case, as shown in Fig. 7, although the ground-truth power of collecting is higher than bucket mapping, it is reversed in Wattch because of the wrong estimation. By learning this pattern, the ML model can realize that, for some consecutive time steps, a power drop in Wattch actually corresponds to a power rise in the real trace. We further illustrate this with a toy example. Assume the number of consecutive time steps $2L+1$ is 3, for time steps t_1, t_2, t_3, t_4, the Wattch-generated power trace of is '17, 17, 15, 15', and the ground truth is '0.50, 0.50, 0.54, 0.54'. During training, the model can learn at t_2 that, the Wattch power trace of '17, 17, 15', corresponds to 0.50 as ground truth. It can also learn at t_3 that, power trace of '17, 15, 15' corresponds to 0.54 as ground truth. In this way, the model learns correlation beyond the single-time-step correlation.

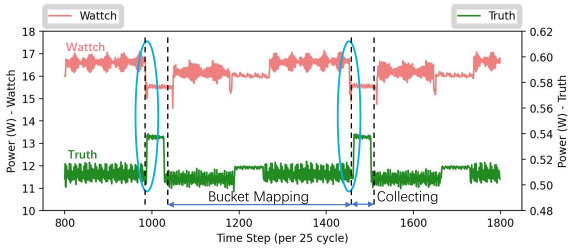


Fig. 7: The comparison between Wattch and ground-truth for workload 'rsort' on configuration C1. The power rises of ground truth in blue circles correspond to power drops in Wattch.

Our fine-grained time-based power trace prediction is tightly coupled with the average power modeling. There are three primary reasons. (1) Time-based power is a natural extension of average power modeling. PANDA's time-based power prediction is achieved by combining two parts, the average power prediction and the fluctuation prediction. The average power prediction is a part of the time-based power prediction. (2) Both time-based and average power modelings are critical for users. From the perspective of user demand, time-based power prediction is critical for modern processor design. (3) Both time-based and average power modelings are essential features for commercial EDA tools. Inspired by these commercial tools, time-based power should be supported besides average power estimation to make PANDA more useful. Therefore, time-based power trace prediction is an important extension to fill the gap between the average power modeling and the standard power estimation tools. Besides, the method of time-based power is also consistent with the average power modeling, because the decoupling of time-based prediction into such two parts is also an analytical operation itself, therefore time-based prediction is a method unifying analytical and ML-based solutions to some extent.

E. Other Design Quality Prediction

In addition to power prediction, the modeling of other design qualities is also supported by PANDA, such as area, performance, and energy. We discuss each design quality below: (1) The area model in PANDA resembles the component-level power model mentioned earlier. However, it only adopts configuration parameters C_i as features, without event parameters E_i. (2) For performance prediction, we find that gem5 exhibits a reasonably accurate correlation denoted by R, but it also displays noticeable absolute errors. To address this, we develop a performance model to calibrate gem5. The model adopts the ratio between the ground truth of the execution cycles and the number of cycles generated by gem5 as the training label. The input features include two parts: all configuration parameters C and selected event parameters E. To capture key performance factors like branch prediction and memory access, we carefully select important event parameters, namely, {numCycles, idleCycles, branchPred condPredicted, branchPred condIncorrect, icache overallMisses, icache ReadReq.mshrMisses, dcache ReadReq.misses, dcache WriteReq.misses, dcache overallMisses, dcache overallMshrMisses}. (3) PANDA enables the evaluation of energy consumption by multiplying its performance prediction with power prediction.

Besides the single design quality, the trade-off between two different design qualities is critical for design evaluation.

Configuration Parameter	C1	C2	C3	C4	C5	C6	C7	C8	C9	C10	C11	C12	C13	C14	C15
FetchWidth	4	4	4	4	4	8	8	8	8	8	8	8	8	8	8
DecodeWidth	1	1	1	2	2	2	3	3	3	4	4	4	5	5	5
FetchBufferEntry	5	8	16	8	16	24	18	24	30	24	32	40	30	35	40
RobEntry	16	32	48	64	64	80	81	96	114	112	128	136	125	130	140
IntPhyRegister	36	53	68	64	80	88	88	110	112	108	128	136	108	128	140
FpPhyRegister	36	48	56	56	64	72	88	96	112	108	128	136	108	128	140
LDQ/STQEntry	4	8	16	12	16	20	16	24	32	24	32	36	24	32	36
BranchCount	6	8	10	10	12	14	14	16	16	18	20	20	18	20	20
MemIssue/FpIssueWidth	1	1	1	1	1	1	1	1	2	1	2	2	2	2	2
IntIssueWidth	1	1	1	1	2	2	2	3	3	4	4	4	5	5	5
DCache/ICacheWay	2	4	8	4	4	8	8	8	8	8	8	8	8	8	8
DTLBEntry	8	8	16	8	8	16	16	16	32	32	32	32	32	32	32
DCacheMSHR	2	2	4	2	2	4	4	4	4	4	4	8	8	8	8
ICacheFetchBytes	2	2	2	2	2	4	4	4	4	4	4	4	4	4	4

TABLE V: The 15 configurations (named C1-C15) used in our experiment. These are divided into 5 domains depending on $DecodeWidth$.

For example, one of the important trade-offs is the trade-off between power and timing, which can be measured by total negative slack (TNS). For a single design, because of different selected synthesis parameters, there will be different synthesis results with different power and timing qualities, causing a power-timing trade-off. Tunable synthesis parameters include max fanout, max capacitance, max transition, high fanout net threshold, set max area, set dynamic optimization, etc. They support trade-offs between low-power and high-performance design. Each synthesis result can be described as a point in the power-versus-timing quality plot. Then the trade-off can be described as the Pareto-optimal curve of all attainable points.

However, predicting the Pareto-optimal curve for an unknown configuration is challenging. Because of limited number of training designs and complexity caused by varying synthesis parameters, directly training a single model to predict multiple points for an unknown configuration is difficult.

To deal with the complexity caused by synthesis parameters, we propose the synthesis-parameters-independent prediction. There are two steps in our method. (1) In the first step, we predict the power and timing of a representative point on the trade-off curve. We choose the point with good qualities in both power and timing as the representative point. Specifically, in this work, we select the point with the minimum multiplication between the absolute value of TNS and Power. Please notice that the prediction for such a single point only depends on the design configuration, so it can be relatively accurate. (2) In the second step, we predict relative positions between other points and the representative point. In this step, we use both the synthesis parameters and configurations as features. Although the relative position depends on synthesis parameters, the relative positions across different configurations have a more similar data distribution than that of the absolute positions. So the prediction can be easier, especially for limited training data. After all of the possible points of a configuration are predicted, we compute Pareto-optimal points and use $TNS = a * ln(b * Power + c)$ to fit a curve, where the a, b, and c are the parameters for the curve. This curve formulation has also been used in the prior work [22].

F. Transferring to New Technology

With the development of the process technology, designers should estimate the power consumption in a new process technology node for an unknown design. But the widely adopted scaling estimation based on $P = CV^2$ is too simple.

To train the ML model for transferring to a new technology node, for each training design, the label is the ratio between power consumption in the target technology node and the source technology node. The features include three parts: (1) The power consumption in the source node predicted by PANDA. (2) The ratio between the scales and voltages of the target and source nodes. (3) The directly-scaled power estimation using the equation $P = CV^2$. We gather ground-truth data for training from only small designs across multiple different technology nodes and train the ML model on them.

IV. Experimental Results

A. Experiment Setup

We generate a dataset based on Chipyard [23]. For a fair comparison with prior works [13], we employed 15 similar RISC-V BOOM [17] CPU configurations in Table V, named C1 to C15, ranging from small to large design sizes. Similar to prior works [14], [16], we further divided these 15 configurations into five domains based on their *DecodeWidth*, which is a key configuration parameter that affects multiple pipeline stages. For power simulation, we used eight workloads in the riscv-tests [24] suite, including dhrystone, median, multiply, qsort, rsort, towers, spmv, and vvadd.

We performed RTL simulation at 1GHz with Synopsys VCS® [25]. The logic synthesis and ground-truth power simulation are performed with Synopsys Design Compiler® [26] and PrimePower [2], respectively. We used the TSMC $40\,\mathrm{nm}$ standard cell library and the corresponding Memory Compiler. In our evaluation of cross-technology node prediction, we also adopted the TSMC $28\,\mathrm{nm}$ and $65\,\mathrm{nm}$ standard cell libraries. These are the most advanced industry libraries accessible to academia. Although ASAP7 [27] is an open-sourced $7\,\mathrm{nm}$ library, it is far from a realistic library and can not provide any power-related information for the SRAM.

To avoid engineers' bias during ML model hyper-parameter tuning, for all these XGBoost models, we simply adopt the default hyper-parameters (i.e., max depth=6, num of estimators=100). PANDA is already sufficiently accurate in this case. For the extension to time-based power trace prediction, we set the length of the time step $T = 25$, and the hyper-parameter $L = 5$, which is the number of consecutive time steps before and after the current time step that the model observes.

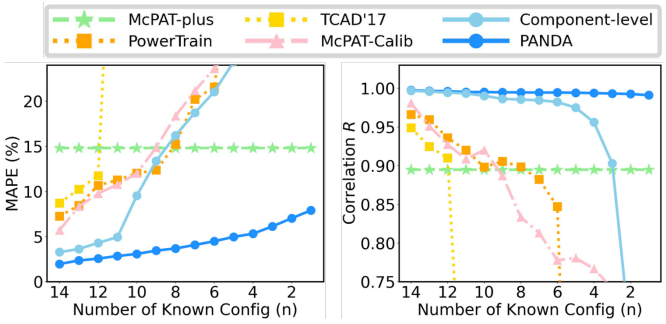


Fig. 8: The MAPE and R of different models under different number of unknown configurations.

B. Summary of Baseline Methods[4]

We compared PANDA with representative prior works, including (a) McPAT (+gem5 [9] [28]) [3], (b) PowerTrain [11], (c) McPAT-Calib [13], (d) TCAD'17 [29]. (e) McPAT-plus, which ideally scales the power prediction of McPAT towards the ground-truth. (f) A weaker variant of PANDA named the Component-level model, as shown in Fig. 4(c). It builds ML models for each component with configuration and event parameters in Table III, but does not adopt resource functions.

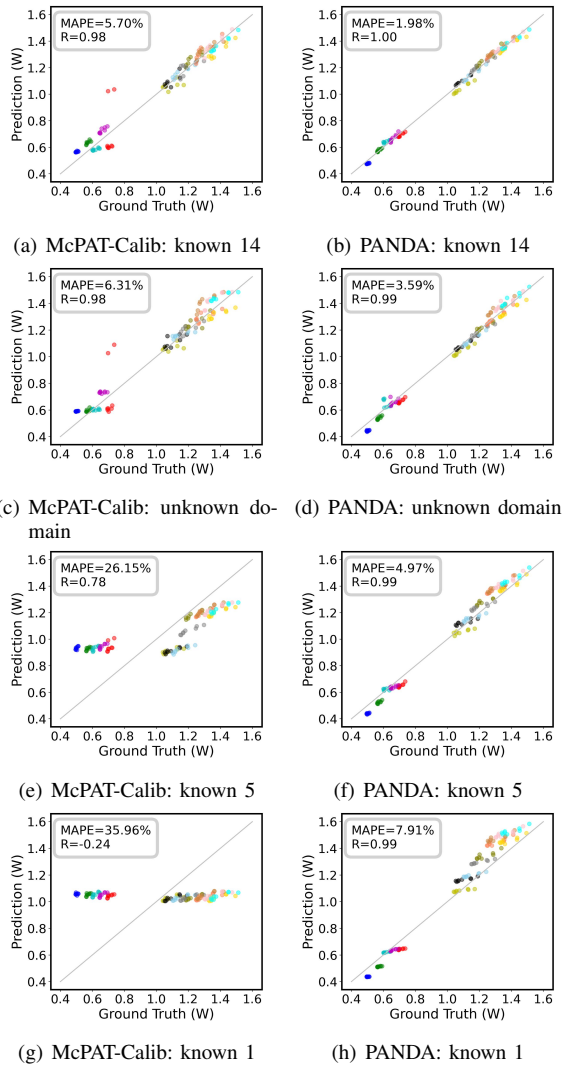
We conduct multiple experiments with different amount of training data for cross-validation. When the number of known configurations for training is n ($n \in [1, 14]$), $15-n$ unknown configurations will be testing data. We evaluate performance with the mean absolute percentage error (MAPE)[5] and correlation coefficient R averaged over all testing configurations.

C. Power Prediction Results

Fig. 8 shows the accuracy of PANDA and our baseline models when they are trained with different number of known configurations. PANDA consistently achieves the lowest MAPE and highest R. The superior performance of PANDA over all ML-based baselines is increasingly obvious as the number of known configurations (i.e., training data amount) in the x-axis decreases. This trend validates PANDA's excellent accuracy given very limited training data.

Here we try to analyze the reasons behind the performance gap between PANDA and representative ML solutions McPAT-Calib [13] and PowerTrain [11] in Fig. 8. In comparison, in previous ML solutions, too much knowledge needs to be learned when training a single ML model from scratch. Also, since they rely on McPAT, they may be limited by McPAT's poor accuracy when approaching a higher accuracy. Another ML baseline TCAD'17 [29] naturally performs poorly since it is designed for on-chip power meter instead of this task.

As for the purely analytical model baseline McPAT-plus, as a correctly-scaled version of McPAT, its accuracy remains unchanged in Fig. 8 regardless of the training data amount. It represents the optimal version of McPAT with ideal scaling parameters. But even with optimal scaling parameters, the accuracy is still relatively low. It indicates that the overall accuracy is limited without ML models. By unifying both analytical and ML techniques, PANDA outperforms it significantly even when there is only one known configuration ($n = 1$) for training. For the original McPAT, while the R is the same as McPAT-plus, the absolute error MAPE is higher than 1000%, so it is not presented in the figure.



(a) McPAT-Calib: known 14 (b) PANDA: known 14

(c) McPAT-Calib: unknown domain (d) PANDA: unknown domain

(e) McPAT-Calib: known 5 (f) PANDA: known 5

(g) McPAT-Calib: known 1 (h) PANDA: known 1

Fig. 9: Accuracy comparison between McPAT-Calib and PANDA under different scenarios when predicting average power. The x-axis is the ground-truth average power, and the y-axis is the prediction of average power.

Finally, the component-level model, as a weaker variant of PANDA with limited architecture knowledge, provides a great decomposition of PANDA's high performance. Its comparison with PANDA can be viewed as a simple ablation study to validate the importance of the resource function proposed in PANDA. By training ML models at the component level, it maintains a reasonable accuracy when the number of known configurations $n > 10$ and outperforms other ML baselines. But PANDA soon outperforms it significantly when training data further decreases. Such a gap shows the contribution of resource functions based on architecture knowledge.

We further visualize predictions in Fig. 9. We compare the strongest baseline McPAT-Calib [13] and PANDA under different scenarios: 1) 14 configurations known; 2) a domain unknown; 3) 5 configurations known; and 4) 1 configurations known. The 'domain unknown' scenario means that 12 config-

[4]We remove detailed description of baseline methods due to page limit. Please kindly refer to our conference version Sec III.B for all details.

[5]MAPE $= 1/n * \sum_{k=1}^{n} |y_k - \hat{y_k}|/y_k$, y_k is label and $\hat{y_k}$ is prediction.

Component Name	14 Known		Unknown Domain		10 Known		5 Known		1 Known		Contribution(%)
	MAPE(%)	R	MAPE(%)	R	MAPE(%)	R	MAPE(%)	R	MAPE(%)	R	
DCache	8.09	0.98	8.62	0.98	13.20	0.94	13.88	0.94	19.85	0.92	11.04
ICache	2.34	1.00	2.55	1.00	3.22	1.00	5.62	1.00	7.40	1.00	14.70
BP	0.96	1.00	1.23	1.00	1.70	1.00	4.40	1.00	6.69	0.99	34.28
RNU	15.64	0.93	16.59	0.90	13.85	0.93	13.33	0.93	16.53	0.93	5.57
I-TLB	0.38	-0.74	0.38	-0.31	0.38	-0.35	0.38	-0.35	0.38	-0.74	0.69
D-TLB	2.17	1.00	2.16	1.00	2.13	1.00	2.12	1.00	2.17	1.00	0.56
Regfile	0.43	1.00	0.44	1.00	0.42	1.00	0.52	1.00	0.70	1.00	6.05
ROB	4.65	1.00	5.76	0.99	4.98	0.99	4.91	0.99	5.15	0.99	3.86
IFU	7.09	0.86	23.51	0.73	16.98	0.75	27.23	0.86	43.43	0.87	8.26
LSU	2.70	1.00	2.15	1.00	2.50	1.00	3.46	1.00	4.79	1.00	3.91
FU Pool	1.39	1.00	2.20	0.99	2.55	0.99	12.35	0.81	17.25	N/A	3.46
ISU	6.77	0.98	8.35	0.98	7.12	0.98	8.34	0.98	9.99	0.98	4.89
Other Logic	4.30	0.97	6.18	0.97	5.65	0.96	6.11	0.95	6.49	0.93	2.73

TABLE VI: The power prediction of each component under different numbers of known configurations and the average percentage contribution of each component to the total power across different configurations and workloads. The correlation metric is the correlation coefficient R.

urations in four domains are known and the 3 configurations in the remaining domain are unknown, then we train the model on the 12 known configurations and test it on the 3 unknown configurations. It is different from 'known 12' because in 'domain unknown' the unknown configurations have a different *DecodeWidth* from all known configurations. The gray line represents the 100% accurate prediction. So the closer the points are to the gray line, the higher the accuracy is. In Fig. 9, each row of sub-figures represents a scenario. For each scenario, PANDA prediction is relatively more concentrated towards the gray line. This reflects the PANDA's consistent advantage over the McPAT-Calib.

From Fig. 9(a)(b) to (g)(h), as known configurations reduce, the accuracy of PANDA remains relatively high, while the accuracy of McPAT-Calib drops significantly. When only one configuration is known, McPAT-Calib predictions become a horizontal line, which means that the model learns almost nothing. Even when known configurations are sufficient (e.g., 14 configurations are known), there are still several McPAT-Calib prediction points with large errors, especially for small designs. We believe the prediction for small designs is more difficult for McPAT-Calib because the similarity between it and its most similar design (for example C1 and C2) is lower than that of the large design (for example C14 and C15). For example, regarding the *FpPhyRegister*, from C1 to C2, it increases by 12, which is 33% of C1. The increase is also 12 from C14 to C15, but it is just 9% of C14. Some other configuration parameters such as *IntPhyRegister*, *DCache/ICacheWay*, and *RobEntry* also demonstrate a similar trend. PANDA can avoid this problem by decoupling the resource function from the power model. In this way, PNADA can leverage all of the known configurations well and can generalize to a large range of configurations, while the McPAT-Calib, a direct tree-based model, can only perform well for data points similar to the known configurations.

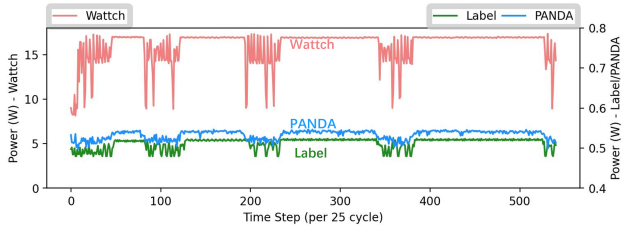
In addition to the total power, we further evaluate PANDA's prediction accuracy in each component. Table VI shows the power prediction accuracy of each component under different numbers of known configurations and the average percentage contribution of each component to the total power across different configurations and different workloads. In PANDA, the accurate prediction of total power relies on the prediction of each component. We can find that PANDA can achieve high accuracy for components with high power consumption. This is the reason for the high accuracy of the total power prediction. More importantly, in practice, computer architects may also require the power of each component for more fine-grained information. Table VI shows that PANDA is not only accurate in total power but also in more fine-grained power values for components, which can be further used to support power optimization better. Similar to the trend of total power, as known configurations are reduced, the accuracy decreases slightly for each component. Especially, the cache is one of the components with high power consumption. In Table VI, for both ICache and DCache, the accuracy remains relatively acceptable even when the known configuration is only one. Another example is the *Other Logic*, which is a complex combination of many smaller components. PANDA can still do well even for such a complex component. But the PANDA does not perform well for I-TLB. It is because the number of I-TLB access in gem5 SE mode is always zero, providing no information for the PANDA. This is consistent with the previous work [13]. For the FU Pool, since its resource function $\boldsymbol{F_{res}} = 1$, it relies on the ML part to learn the pattern from training samples. When the known configuration is only one, almost all predictions are similar to the power of this single training sample. In this special case, the correlation R of FU Pool is not a valid metric, thus denoted as N/A in Table VI. However, the I-TLB and FU Pool consume a small percentage of the total power, so the inaccurate prediction does not have an obvious negative impact on the total power prediction.

D. Prediction of Time-based Power Trace

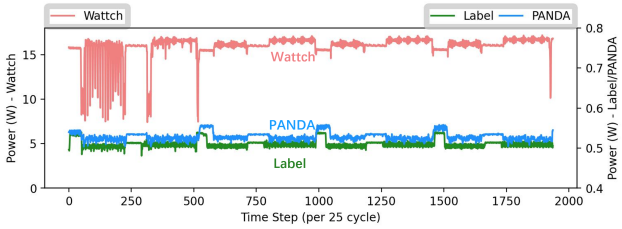
In our experiment of time-based power trace prediction, we focus on the small configurations C1 to C6. Compared with large configurations C7 to C15, these small configurations naturally have relatively low power consumption. So these small configurations are more similar to low-power mobile processors than large configurations. Therefore, we choose them for our experiment. Moreover, accurate per-cycle power simulation on smaller cores is also much faster than large ones, making data generation feasible in weeks. To clearly evaluate time-based power, the workloads for the experiment better have obvious fluctuation patterns. So we adopt 5 workloads with clear fluctuations, including multiply, rsort, vvadd, multiply-acc, and div-acc. Multiply-acc and div-acc are derived from multiply and div by inserting some accumulation loop. By doing so, the pattern in the power trace can be

Config	Max(%)							Min(%)							MAPE(%)						
	Wattch	Wattch-plus	McPAT	McPAT-plus	Powertrain	LSTM	Ours	Wattch	Wattch-plus	McPAT	McPAT-plus	Powertrain	LSTM	Ours	Wattch	Wattch-plus	McPAT	McPAT-plus	Powertrain	LSTM	Ours
C0	3175	2.35	1454	29.14	32.65	55.99	4.67	1562	48.25	1550	37.08	31.63	5.88	4.70	2982	6.62	1473	30.67	26.87	15.66	4.42
C1	3508	12.30	1273	9.66	19.39	90.31	4.47	1834	39.80	1365	16.94	2.86	20.40	5.74	3329	10.22	1272	13.54	6.81	31.70	4.90
C2	3672	17.41	1078	9.41	142.1	31.89	1.85	1831	39.90	1113	6.74	164.4	92.43	3.88	3543	14.42	1025	13.54	151.8	49.22	2.50
C3	3718	18.85	1157	1.54	8.61	39.81	7.00	1927	36.90	1232	4.16	18.50	21.56	8.45	3464	12.68	1143	3.29	10.82	25.53	6.63
C4	3432	9.94	1449	26.75	80.56	68.18	2.58	1781	41.44	1584	37.78	72.70	9.17	5.66	3236	7.42	1459	27.55	78.94	15.75	3.70
C5	2224	27.66	875.7	28.81	43.51	253.3	3.48	1131	61.67	900.7	26.99	55.70	187.6	2.93	2064	32.65	823.3	32.63	47.50	222.4	3.83
Average	3289	14.75	1215	17.55	54.48	89.92	4.10	1670	44.66	1291	21.69	57.64	56.17	5.13	3104	14.00	1199	19.54	53.80	60.05	4.34

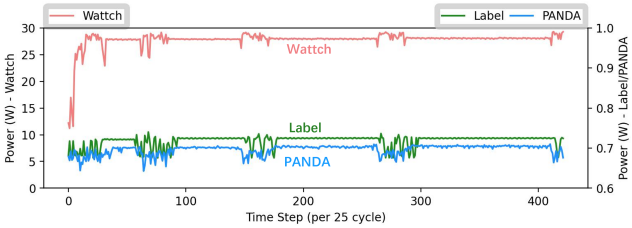
TABLE VII: The comparison of power trace prediction with prior works. This table shows three metrics related to the value range: maximum power, minimum power, and the MAPE of the time-based power trace. Each data is the average value across different workloads.



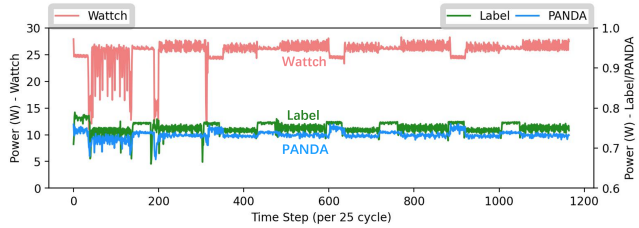
(a) The power trace of div-acc on C1



(b) The power trace of rsort on C1



(c) The power trace of div-acc on C3



(d) The power trace of rsort on C3

Fig. 10: The visualized comparison of power trace prediction of PANDA, Wattch, and ground truth. The left axis is the axis of Wattch, the right axis is the axis of PANDA and ground truth. It shows that the prediction of PANDA is much more similar to the ground truth.

easy to observe. And we use the Wattch and Wattch-plus, the Wattch scaled towards the ground truth, as our baseline. In this experiment, we use 5 configurations with all workloads for training and 1 configuration with all workloads for testing.

In Table VII, we evaluate three important metrics for time-based power of each workload, including peak power, minimum power, and average per-time-step error for each time step, which can be measured by MAPE. The shown data is average value for each configuration across different workloads. The peak and minimum power are critical for CPU design because they reflect the two extreme conditions to be supported at runtime. The average per-time step error reflects power model accuracy in general conditions. Besides Wattch and Wattch-plus, we also compare PANDA with three extra baselines, McPAT, McPAT-plus, and Powertrain, with small modifications to collect events every time step. We evaluate two models we proposed, XGBoost (denoted as PANDA) and LSTM. The result shown in Table VII verifies that PANDA outperforms others in peak power, minimum power, and average per-time step error for different configurations. We can also find that XGBoost can outperform LSTM obviously, with MAPE of 4% and 60% respectively. The poor accuracy of LSTM is largely due to overfitting when training data is limited. In conclusion, we adopt XGBoost as our ML model.

Fig. 10 visualizes the prediction result and the comparison with Wattch's power trace. Due to the page limit, we only show the results of two workloads div-acc and rsort on two configurations C1 and C3 respectively. The left axis is the power trace of the Wattch, and the right is the prediction of PANDA and ground truth. We can find that the power trace of Wattch is far from the ground truth, although there are slight similarities in their patterns. PANDA is significantly better than the Wattch. For example, in the workload 'rsort', there are three obvious power rises in the ground truth. For the Wattch, in all corresponding three time steps, it predicts power drops. In comparison, PANDA correctly predicts power rises, with a similar pattern as the ground truth.

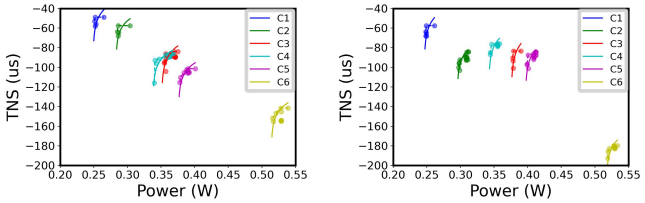
Design Quality	Baseline			PANDA	
	Baseline Method	MAPE(%)	R	MAPE(%)	R
Area	McPAT-plus	12.90	0.98	2.92	0.99
Performance	gem5	26.79	0.98	6.69	0.98
Energy	gem5 + McPAT-Calib	31.87	0.97	9.51	0.98

TABLE VIII: The comparison of Area, Performance, and Energy prediction between baseline and PANDA

As an architecture-level model, PANDA has low runtime and memory overhead compared with the traditional VLSI flow. The actual runtime and memory overhead depend on the software implementation and hardware resources. Conservatively, based on our measurement, the runtime of time-based prediction of 6 configurations for 8 workloads is within 2 minutes, which is mainly spent on Wattch, and the model only takes less than 30 seconds. In comparison, the corresponding VLSI flow takes about 3 days. Regarding the memory overhead, our method consumes tens of MB which is the memory requirement of Wattch, while the VLSI flow takes tens of GB.

E. Prediction of Other Design Qualities

In addition to the power prediction, Table VIII shows the prediction accuracy of PANDA on area, performance, and energy, adopting the 'unknown-domain' scenario. Since most



(a) The ground truth trade-off curve for dhrystone (b) The predicted trade-off curve for dhrystone

Fig. 11: The prediction of Pareto-optimal curve of each configuration. The left subfigure shows ground truth and the right subfigure shows predictions. Pareto-optimal curves show the trade-off between power and timing. The dashed curve is the actual Pareto-optimal curve observed from the points. The solid curve is fitted with the Pareto-optimal points. We only plot the points close to the optimal curve.

prior power models do not cover these design qualities, in this section, we use McPAT-plus, the McPAT scaled towards the ground truth, as the baseline for area estimation, and gem5 as the baseline to estimate performance, measured with the number of cycles for each workload. For the energy, we multiply the performance generated by gem5 by the power predicted by McPAT-Calib as the energy baseline. For baselines, we observe a reasonable correlation but a huge absolute error value. In comparison, PANDA can even achieve higher correlation while keeping the MAPE error under 10%.

The prediction of Pareto-optimal curve for TNS-power trade-off is evaluated on six configurations from C1 to C6. To demonstrate the trade-off, in our experiment, we set a high clock frequency for synthesis, thus the TNS is always negative. To collect the ground truth of the pareto curve, we modify the parameters adopted in synthesis to demonstrate the trade-off between TNS and power. Based on the netlist we can get the TNS. Then we perform power simulations for each netlist and the average power across different workloads is adopted as the power in the TNS-power trade-off figure. Based on these points, we can get the pareto curve. Fig. 11 illustrates the prediction of each configuration with *dhrystone* workload, where only points near pareto curve are shown. Prediction with other workloads is similar since we observe that the design quality trade-off mainly depends on design configurations and synthesis parameters. Table IX shows the accuracy of our prediction for each workload, where the MAPE is calculated with the average power and TNS for Pareto-optimal points in the ground truth and our prediction. In Fig. 11, the left subfigure is ground truth, and the right one shows prediction. We analyze the result from two aspects, 1) the absolute position of the representative point; 2) the relative position of other points, as discussed in the methodology. The representative point prediction can be reflected by the position of each Pareto-optimal curve. We observe that the representative point prediction is accurate, reflecting accurate single-design-quality prediction in PANDA. The prediction for relative positions of other points can be evaluated by comparing the shape of the trade-off curve. We also observe that the shape of the predicted Pareto-optimal curve is similar to ground truth. In summary, PANDA can predict design trade-offs well at the architecture level.

Workload	dhrystone	median	multiply	qsort	rsort	towers	spmv	vvadd
MAPE for Ave TNS (%)	22.6	19.8	18.8	20.9	20.9	18.0	19.3	18.3
MAPE for Ave Power (%)	2.6	4.2	2.6	1.9	2.1	2.0	3.5	2.7

TABLE IX: The accuracy of the TNS-power trade-off prediction for each workload. It is calculated by average power and TNS for Pareto-optimal points in the ground truth and our prediction.

Source	Target	MAPE-Original(%)	MAPE-Scaled(%)	MAPE-PANDA(%)
28 nm	40 nm	51.51	30.98	14.83
28 nm	65 nm	73.12	40.42	10.16
40 nm	28 nm	115.98	20.03	6.24
40 nm	65 nm	43.39	10.07	11.66
65 nm	28 nm	289.02	25.52	5.28
65 nm	40 nm	83.94	5.91	14.24
Average		109.49	22.16	10.40

TABLE X: Cross-technology prediction by 1) prediction at source technology, 2) directly-scaled prediction towards target technology, 3) prediction transferred to target technology by PANDA.

F. Cross-Technology Prediction

To verify the cross-technology prediction in PANDA, we conduct the experiment with three technology nodes: TSMC 28 nm 0.8 V, TSMC 40 nm 1.1 V, and TSMC 65 nm 1.2 V. PANDA's ML transferring model is trained using approximately 20 small designs synthesized across these three technologies. These small designs, on average, consist of only thousands of gates, while the smallest configuration C1 comprises 0.3 million gates. The transfer model predicts the power of an unknown design configuration at the target node based on PANDA's prediction at the source node.

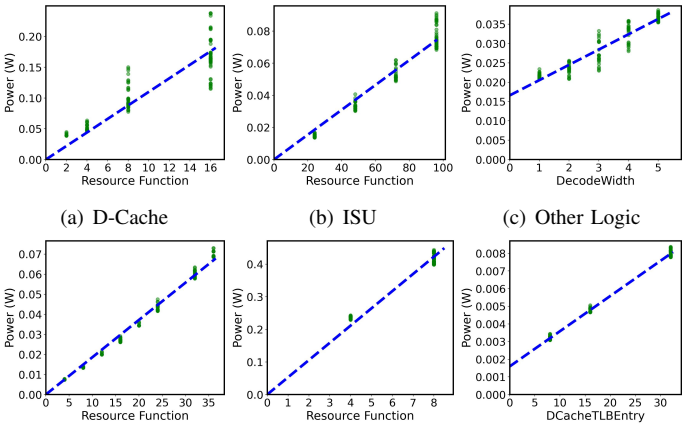
We compare three types of predictions in Table X, including 1) the original prediction based on the PANDA's power model at the source technology node; 2) the directly-scaled prediction to the target node based on CV^2; 3) the transferred prediction of our proposed ML transferring model. The average MAPE for the scaled prediction is 22.16%, while for our model, it is 10.40%. These results show that the prediction accuracy of PANDA can outperform direct scaling significantly.

V. Discussion

In this section, we provide an additional analysis for the reasonability of the PANDA and discuss the potential integration of PANDA. We first visualize the correlation between ground-truth power and the resource function in Sec. V-A, and then discuss the correctness from the data distribution aspect in Sec. V-B. Moreover, we finally discuss the potential integration to other architecture-level tools in Sec. V-C.

A. Correlation between Power and Resource Function

To further validate the correctness of PANDA, in Fig. 12, we visualize the correlation between each component's power and its resource function $\boldsymbol{F}^i_{\boldsymbol{res}}$ respectively. We only show 6 representative components here because of page limit. We categorize components into two classes. The first class is component without 'bias' in resource function, including D-Cache, ISU, LSU, and BP. We observe that their power mainly scales proportionally with resource function. The second class is component with 'bias' in resource function, including D-TLB and Other Logic. For Other Logic, we show *DecodeWidth*



(a) D-Cache (b) ISU (c) Other Logic

(d) LSU (e) BP (f) D-TLB

Fig. 12: Component power vs its resource function $\boldsymbol{F_{res}^i}$. For the Other Logic part in subfigure (c) and D-TLB part in subfigure (f), we exclude the bias term from its resource function in the x-axis to indicate the correlation.

in x-axis, and then we can find its power approximately proportional to the resource function (i.e., *DecodeWidth* + bias). D-TLB also shows a similar trend.

In conclusion, the correlation shown in Fig. 12 implies the correctness of proposed resource functions $\boldsymbol{F_{res}^i}$, which will be multiplied by ML model output to generate the final power prediction of each component, as defined by Equation (5).

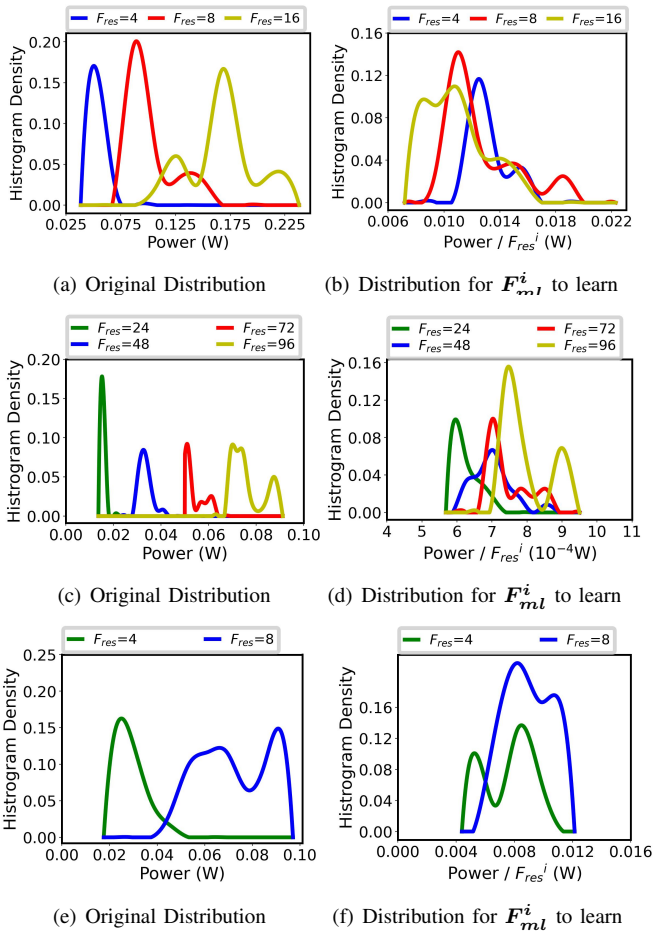
B. PANDA Analysis: Data Distribution Aspect

From another perspective, Fig. 12 also shows that for the same resource function value in x-axis, there are still many power value variations in the vertical direction, caused by the difference in other configuration parameters and event parameters. These variations will be captured by PANDA's ML part of each component $\boldsymbol{F_{ml}^i}$. According to Equation (5), the ML model $\boldsymbol{F_{ml}^i}$ is actually trained to predict the power divided by the resource function (i.e., power/$\boldsymbol{F_{res}^i}$).

To analyze such vertical power variation for different resource function values, we visualize the original power distribution of D-Cache, ISU, and BP in Fig. 13(a)(c)(e) respectively. It corresponds to vertical points in Fig. 12(a)[6](b)(g). Then we further visualize the distribution of power/$\boldsymbol{F_{res}^i}$ in Fig. 13(b)(d)(f). This is what the ML model is required to learn. The comparison between Fig. 13(a)(c)(e) and Fig. 13(b)(d)(f) shows an interesting pattern and provides another explanation of the superior performance of PANDA.

In detail, as shown in Fig. 13(a)(c)(e), the power distributions of configurations with different resource function values are largely different. As a result, when training data is limited, ML models may only see training samples from a few distributions, so perform badly on testing designs that are from unknown other distributions. The gap between different distributions of original power is large, causing an obvious prediction error. In comparison, PANDA actually trains the ML model to predict power/$\boldsymbol{F_{res}^i}$, as shown in Fig. 13(b)(d)(f). This power/$\boldsymbol{F_{res}^i}$ objective provides obviously more similar distributions compared with the original power distribution.

[6]Configurations with $F_{res} = 2$ only account for 6% among all configurations of D-Cache displayed in Fig. 12(a). Therefore, we discard this small part in Fig. 13, only showing $F_{res} = 4, 8, 16$.



(a) Original Distribution (b) Distribution for $\boldsymbol{F_{ml}^i}$ to learn

(c) Original Distribution (d) Distribution for $\boldsymbol{F_{ml}^i}$ to learn

(e) Original Distribution (f) Distribution for $\boldsymbol{F_{ml}^i}$ to learn

Fig. 13: Power distribution of key components: (a)(b) D-Cache; (c)(d) ISU; (e)(f) BP. They correspond to the power distribution of points in Fig. 12. Sub-figures (a)(c)(e) describe the original power, learned by existing ML methods. Sub-figures (b)(d)(f) describe the power divided by resource function (i.e., power/$\boldsymbol{F_{res}^i}$), learned by PANDA's ML model. PANDA's ML part $\boldsymbol{F_{ml}^i}$ learns more similar distributions, benefiting accuracy when training data is limited.

Even when training data is limited, the ML part's prediction will fall into a similar distribution anyway, without causing a large error. This analysis provides one more rationale for our multiplying ML model with resource function in PANDA.

C. Potential Integration

PANDA can be potentially integrated with other architecture-level physical quality models. There are some thermal modeling tools such as HotSniper [30], HotSpot [31], and PACT [32], and architecture-level hotspot estimators such as HotGauge [33]. First, these tools are based on the McPAT, where the power estimation of McPAT is the input of these tools. For example, the power estimation of each component such as the core and the cache can be estimated by McPAT, then the thermal modeling tools can use it to estimate the temperatures. Therefore, our power model can naturally improve the accuracy of these tools. Besides, the McPAT is usually used as a standalone module to provide power estimation, so we think replacing the McPAT with PANDA is easy to implement. Second, the modeling methods of these thermal modeling tools and hotspot estimators are also natively inaccurate. Therefore, besides only taking a more accurate power estimation as input, to further improve

the accuracy, the method proposed in PANDA that unifies the analytical and ML-based model can be potentially integrated into these tools. Architecture-level analysis for the target physical quality such as thermal and hotspot can be utilized to capture simple but important patterns, and a data-driven model can further learn more complex patterns with a few training data. Moreover, CACTI [34] is an architecture-level model for memory system, such as cache. With the analysis of how each operation in memory system consumes energy, we can also integrate our method to improve CACTI. These points can be our future work.

VI. Conclusion

In this work, we introduce PANDA, a comprehensive architecture-level CPU modeling framework for power and other design qualities. PANDA unifies analytical and ML approaches to model the average power. PANDA also extends the average power model to support time-based power modeling. Besides power, other design qualities are also supported. It can also evaluate TNS-power trade-off derived from different synthesis parameters, accurately predicting the Pareto-optimal curve. PANDA enables early-stage evaluation of CPU, serving as a valuable addition to the toolkit of architects.

References

[1] Siemens, "PowerPro RTL Low-Power," 2023. [Online]. Available: https://www.mentor.com/hls-lp/powerpro-rtl-low-power/
[2] Synopsys, "PrimePower: RTL to signoff power analysis," 2023. [Online]. Available: https://www.synopsys.com/implementation-and-signoff/signoff/primepower.html
[3] S. Li *et al.*, "McPAT: An integrated power, area, and timing modeling framework for multicore and manycore architectures," in *MICRO*, 2009.
[4] A. Tang *et al.*, "Mcpat-pvt: Delay and power modeling framework for finfet processor architectures under pvt variations," *IEEE TVLSI*, 2014.
[5] A. Guler and N. K. Jha, "Mcpat-monolithic: An area/power/timing architecture modeling framework for 3-d hybrid monolithic multicore systems," *IEEE TVLSI*, 2020.
[6] D. Brooks *et al.*, "Wattch: A framework for architectural-level power analysis and optimizations," *ACM SIGARCH Computer Architecture News*, 2000.
[7] S. L. Xi *et al.*, "Quantifying sources of error in mcpat and potential impacts on architectural studies," in *HPCA*. IEEE, 2015.
[8] T. Nowatzki *et al.*, "Architectural simulators considered harmful," *IEEE Micro*, 2015.
[9] N. Binkert *et al.*, "The gem5 simulator," *ACM SIGARCH computer architecture news*, 2011.
[10] T. E. Carlson *et al.*, "Sniper: Exploring the level of abstraction for scalable and accurate parallel multi-core simulation," in *SC*, 2011.
[11] W. Lee *et al.*, "Powertrain: A learning-based calibration of mcpat power models," in *ISLPED*. IEEE, 2015.
[12] M. Sagi *et al.*, "A lightweight nonlinear methodology to accurately model multicore processor power," *IEEE TCAD*, 2020.
[13] J. Zhai *et al.*, "McPAT-Calib: A RISC-V BOOM microarchitecture power modeling framework," *IEEE TCAD*, 2022.
[14] J. Zhai *et al.*, "Microarchitecture power modeling via artificial neural network and transfer learning," in *ASPDAC*, 2023.
[15] B. C. Lee and D. M. Brooks, "Illustrative design space studies with microarchitectural regression models," in *HPCA*. IEEE, 2007.
[16] C. Bai *et al.*, "BOOM-Explorer: RISC-V BOOM microarchitecture design space exploration framework," in *ICCAD*. IEEE, 2021.
[17] J. Zhao *et al.*, "Sonicboom: The 3rd generation berkeley out-of-order machine," in *CARRV*, 2020.
[18] Q. Zhang *et al.*, "Panda: Architecture-level power evaluation by unifying analytical and machine learning solutions," in *ICCAD*. IEEE, 2023.
[19] J. H. Friedman, "Greedy function approximation: a gradient boosting machine," *Annals of statistics*, 2001.
[20] T. Chen and C. Guestrin, "Xgboost: A scalable tree boosting system," in *KDD*, 2016.
[21] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, 1997.
[22] P. Sengupta *et al.*, "How good is your Verilog RTL code? a quick answer from machine learning," in *ICCAD*, 2022.
[23] A. Amid *et al.*, "Chipyard: Integrated design, simulation, and implementation framework for custom socs," *IEEE Micro*, 2020.
[24] *RISC-V Tests*, https://github.com/riscv-software-src/riscv-tests, 2022.
[25] "VCS® functional verification solution," https://www.synopsys.com/verification/simulation/vcs.html, 2021.
[26] "Design Compiler® RTL Synthesis," https://www.synopsys.com/implementation-and-signoff/rtl-synthesis-test/design-compiler-nxt.html, 2021.
[27] L. T. Clark *et al.*, "Asap7: A 7-nm finfet predictive process design kit," *Microelectronics Journal*, 2016.
[28] A. Roelke and M. R. Stan, "Risc5: Implementing the risc-v isa in gem5," in *CARRV*, 2017.
[29] M. J. Walker *et al.*, "Accurate and stable run-time power modeling for mobile and embedded cpus," *IEEE TCAD*, 2016.
[30] A. Pathania and J. Henkel, "Hotsniper: Sniper-based toolchain for many-core thermal simulations in open systems," *IEEE Embedded Systems Letters*, 2018.
[31] W. Huang *et al.*, "Hotspot: A compact thermal modeling methodology for early-stage vlsi design," *IEEE TVLSI*, 2006.
[32] Z. Yuan *et al.*, "Pact: An extensible parallel thermal simulator for emerging integration and cooling technologies," *IEEE TCAD*, 2021.
[33] A. Hankin *et al.*, "Hotgauge: A methodology for characterizing advanced hotspots in modern and next generation processors," in *IISWC*, 2021.
[34] N. Muralimanohar *et al.*, "Cacti 6.0: A tool to model large caches," *HP laboratories*, 2009.

Qijun Zhang received the B.Eng. degree from Tongji University, Shanghai, China, in 2022. He is currently a Ph.D. student in the Department of Electronic and Computer Engineering (ECE) at the Hong Kong University of Science and Technology (HKUST). His research interests include Computer Architecture and Electronics Design Automation.

Mengming Li received the M.Sc degree from the School of Software Technology, Zhejiang University, Hangzhou, China, in 2023. He is currently a Ph.D. student in the Department of Electronic and Computer Engineering (ECE) at the Hong Kong University of Science and Technology (HKUST). His research interests include Computer Architecture and Electronics Design Automation.

Andrea Mondelli received his PhD degree in Computer Architecture from INRIA under the supervision of prof. Andre Seznec. Over the past 20 years he has worked in the fields of computer architecture, memories, and cybersecurity in Italy, the USA, France, China, and the UK, publishing multiple manuscripts, conference papers, and a book on memory coherence protocols. He was part of RISC-V International, co-chairing the Virtual Memory group. His research interests include high-performance and low-power CPU design.

Zhiyao Xie is an Assistant Professor of the Department of Electronic and Computer Engineering (ECE) at the Hong Kong University of Science and Technology (HKUST). Zhiyao received his Ph.D. degree from Duke University in 2022 and B.Eng. from City University of Hong Kong in 2017. His research interests include machine learning algorithms for EDA and VLSI design. He has received multiple prestigious awards, including the IEEE/ACM MICRO 2021 Best Paper Award, ACM SIGDA SRF Best Research Poster Award 2022, ASP-DAC 2023 Best Paper Award, ACM Outstanding Dissertation Award in EDA 2023, EDAA Outstanding Dissertation Award 2023, and the 2023 Early Career Award from Hong Kong Research Grants Council (RGC).