PANDA: Architecture-Level Power Evaluation by Unifying Analytical and Machine Learning Solutions

Qijun Zhang[1], Shiyu Li[2], Guanglei Zhou[2], Jingyu Pan[2], Chen-Chia Chang[2], Yiran Chen[2], Zhiyao Xie[1*]
[1]Hong Kong University of Science and Technology, [2]Duke University
qzhangcs@connect.ust.hk, {shiyu.li, guanglei.zhou, jingyu.pan, chenchia.chang, yiran.chen}@duke.edu, eezhiyao@ust.hk

***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 data-independent analytical power models like McPAT and Wattch have been criticized for their unreliable accuracy. While some machine learning (ML) methods have been proposed for architecture-level power modeling, they rely on sufficient known designs for training and perform poorly when the number of available designs is limited, which is typically the case in realistic scenarios.**

In this work, we derive a general formulation that unifies existing architecture-level power models. Based on the formulation, 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, which is a common challenge in practice. Besides being an excellent power model, it can predict area, performance, and energy accurately. PANDA further supports power prediction for unknown new technology nodes. In our experiments, besides validating the superior performance and the wide range of functionalities of PANDA, we also propose an application scenario, where PANDA proves to identify high-performance design configurations given a power constraint.

I. Introduction

Power efficiency is a critical design objective in modern microprocessor design. With the continuous growth in chip complexity, optimizing designs for better power efficiency requires a significant amount of manpower and long turnaround time. Therefore, there is a high demand for fast, yet high-fidelity early-stage power modeling techniques to facilitate efficient design optimizations. For instance, chip architects may need to efficiently evaluate the power efficiency, as well as other design qualities of multiple new design configurations at the architecture-level, before starting the detailed register-transfer level (RTL) implementation and subsequent VLSI design flow.

However, traditional power modeling techniques fall short of meeting the requirements. The existing standard VLSI design flow generates accurate power evaluations through multiple design stages, including RTL implementation, logic synthesis, RTL simulation with realistic workloads, and gate-level power simulation using commercial tools [1], [2]. Unfortunately, this process is excessively time-consuming for evaluating each architectural-level design configuration. As for faster alternatives, widely-adopted architectural-level power models such as McPAT [3]–[5], and Wattch [6] have been criticized for their unreliable accuracy, as discussed in many prior studies [7], [8]. Despite some improvements in subsequent works, they are primarily developed in-house to cater to proprietary designs [7].

In recent years, some ML methods [9]–[11] were proposed to directly calibrate the existing analytical models like McPAT [3] (i.e., using McPAT output as ML models' input). In this way, they may generate a more accurate estimation, when the target design architecture is similar to already known designs in the training dataset. However, their accuracies degrade significantly when applied to unknown new design configurations. This problem is particularly severe when training data is limited, which is frequently the case in practice. As mentioned, the label collection of a new sample (i.e., new architecture-level configuration) requires the actual implementation of it through VLSI flow and workload-based simulations. This whole process can be extremely time-consuming. A most recent ML work [10] proposes to predict unknown new designs with transfer learning. But it still requires a few ground-truth samples in the target configuration domain, which is still time-consuming to generate in practice. In addition, some design space exploration (DSE) works [12], [13] also develop their own ML-based power models, which are trained iteratively based

[*]Corresponding Author

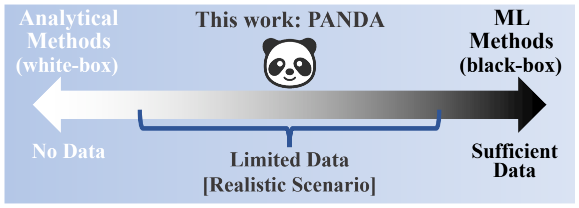

Analytical Methods (white-box)
This work: PANDA
ML Methods (black-box)
No Data
Limited Data [Realistic Scenario]
Sufficient Data

Fig. 1: A summary of architecture-level power modeling methods. Traditional analytical methods adopt inaccurate oversimplified handcrafted model, while the accuracies of ML methods degrade significantly when applied to unknown new configurations. PANDA unifies both analytical and ML solutions, addressing their long-lasting limitations.

on labels collected during exploration. Besides still being limited by training data, these models also typically cannot incorporate workload-related information, thus failing to predict vector-based power values for each specific workload.

In this work, we first propose our qualitative analysis of both analytical [3], [6] and ML-based [9]–[13] architecture-level models, as summarized in Fig. 1. When there is sufficient training data that can cover the potential testing data the model will encounter, the ML model performs better. However, when very limited data is available, ML models can even be misleading, and traditional analytical methods become the only option. Nevertheless, in realistic scenarios, most design teams are somewhere in the middle, with a very limited number of already-implemented architecture configurations that can be used as training data. The high demand for the training data amount and diversity sets a high barrier to the wide adoption of ML solutions [9]–[13] in practice.

Inspired by these observations, we propose an innovative new architectural-level power modeling solution named PANDA. As Fig. 1 shows, its name PANDA implies unifying *white*-box analytical models and *black*-box ML models[1], benefiting from the complementary advantages of both sides. PANDA adopts an analytical framework to model the hierarchy of individual components. For each component, it integrates an ML model with a simple customized analytical function, which is based on identified key configuration parameters of this component. In this way, the analytical function captures the key pattern of the component, leaving more complex detailed patterns to be learned by the ML portion. As a result, PANDA significantly outperforms state-of-the-art solutions, especially when training data is very limited. The low requirement on training data makes PANDA readily to be widely adopted. In addition, unlike most existing ML methods [9]–[13] that rely on existing analytical models like McPAT [3] as model input features and simply perform calibration, PANDA is a standalone solution without relying on any existing analytical models.

Besides power modeling, to comprehensively evaluate a design at the architecture level, evaluations 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 [14] are widely-adopted, but it is not sufficiently accurate. For gate area, McPAT [3] is widely-adopted but inaccurate. To solve these challenges, our solution PANDA also enables accurate evaluations of

[1]Please notice that most existing ML works [9]–[13] directly adopt analytical model McPAT's output as an input feature and perform calibration with ML. But for the main part of their method, unlike analytical solutions, they do not explicitly capture/encode any patterns based on architecture knowledge. This is essentially different from how PANDA integrates analytical model, and thus we categorize these prior works [9]–[13] as ML methods in Fig. 1.

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 I: Our identified architecture-level design configuration parameters C_i and event parameters E_i of each ith component.

other design objectives.

The key contributions in this work 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[2], which unifies these two major types of methods. Given different training dataset sizes, it outperforms state-of-the-art baselines by 5% to 30% in error percentage. The gap is increasingly obvious as the training data amount decreases.
- In this work, we provide a unified formulation that can express all existing architecture-level power models including PANDA. It helps to demonstrate the portion of analytical and ML techniques in each method and guide the accuracy analysis when training data amount varies.
- Unlike most ML methods that directly calibrate McPAT, PANDA does not rely on any existing analytical power model like McPAT to provide features. Instead, PANDA develops its own simpler analytical function for each component based on architecture knowledge, leaving more complex patterns to be learned by the ML part. Such a standalone solution avoids propagating the unreliable accuracies of McPAT.
- Besides power modeling, PANDA also models other design objectives including design performance, area, and energy.
- Finally, PANDA further supports the power prediction targeting unknown new technology nodes.

II. Methodology

A. CPU Components and Configuration Parameters Identification

For our target modern out-of-order (OoO) CPU core[3], all major architecture-level configuration parameters and event parameters explored in this work are listed in Table I. Denote all these configuration parameters as a set C and all workload-related architecture-level event parameters as E. To develop PANDA, we further identified key CPU components that can be individually modeled in power evaluation. Table I lists our identified configuration parameters and event parameters related to each component, and Fig. 2 shows the overall CPU architecture including these components.

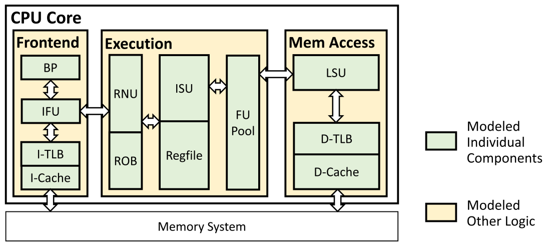


Fig. 2: The architecture of our target Out-of-Order RISC-V CPU core. The green blocks indicate key individual components modeled by PANDA. These components correspond to the Table I. The yellow block refers to the Other Logic indicated in Table I.

As the CPU core architecture in Fig. 2 shows, all our modeled individual components can be categorized into three main parts: frontend, execution, and memory access, with each part comprised of several key components, as introduced below.

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

All other CPU design logic not covered by above components is referred to as Other Logic.

Generally, assuming there are N components in the target CPU design, our identified configuration parameters related to the ith component are denoted as C_i, with $C = \{C_i \,|\, i \in [1, N]\}$. Similarly, denote the architecture-level event parameters related to this component as E_i, with $E = \{E_i \,|\, i \in [1, N]\}$. Both C_i and E_i of each component can be simply looked up in Table I.

B. A General Formulation of Existing Power Modeling Methods

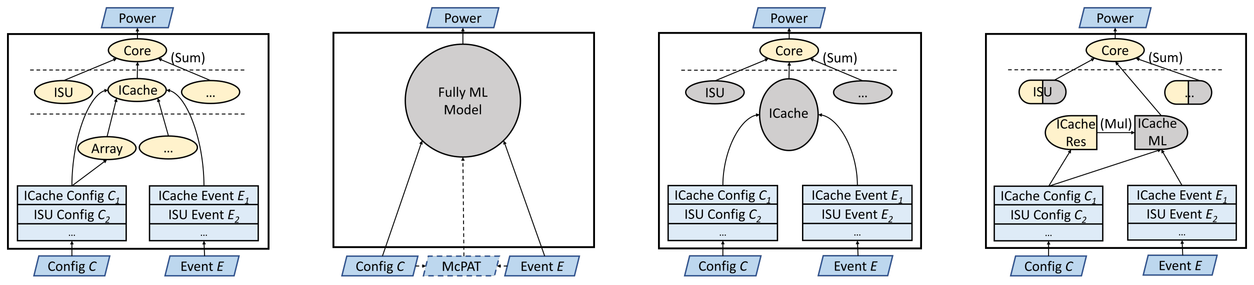
Fig. 3 further provides conceptual visualizations of existing analytical power models [3], [6], ML-based models [9]–[11], and PANDA, under a similar framework, with all CPU configuration parameters C and workload-related event parameters E as model input candidates. We find that power modeling works can be expressed as a unified formulation. So we first propose our formulation of the two types of prior works, then we will start introducing our new method in the next Subsection.

Formulation of ML works. Existing ML solutions [9]–[11] simply build ML models targeting total power values, based on all available design configuration parameters C and event parameters E, as shown below[4]. The $\boldsymbol{F_{ml}}$ denotes data-driven ML methods, including data-

[2]It has been open-sourced at https://github.com/zqj2333/PANDA

[3]PANDA experiments on the RISC-V OoO CPU core BOOM [15]. It can be extended to other CPU types with minor modifications.

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



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

Fig. 3: Illustration of different power modeling methods, yellow means the analytical parts and dark means the ML parts. (a) The analytical method [3], [6]. (b) The ML method [9]–[13], the method mainly relies on ML model. (c) Component-level power model, a much weaker variant of PANDA for ablation study. It builds an ML model to predict the power of each component. (d) PANDA, the model of each component consists of two sub-model, the yellow one means the resource function, which is analytical, while the dark one means the ML model.

driven feature selection and the development of ML models. It can be formulated below, with P_{ml} denoting the power prediction value.

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

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

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

It means existing ML methods adopt the available configuration parameters and event parameters information from all components to evaluate the total power value 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,

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

where an analytical 'resource function' $\boldsymbol{F^i_{res}}(C_i)$ first calculates an intermediate value that reflects the resource consumption corresponding to the configuration parameters. Then another 'aggregation function' $\boldsymbol{F^i_{agg}}$ combines both resource values $\boldsymbol{F^i_{res}}(C_i)$ and event parameters E_i to calculate the power of this component for each workload.

We illustrate the aforementioned analytical methods with the actual 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 consumption 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.

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

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.

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

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

C. The General Formulation of 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.

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

Similar to the notations used in Equation (1)(2), here the $\boldsymbol{F^i_{agg}}$ and $\boldsymbol{F^i_{res}}$ denote analytical functions, and $\boldsymbol{F^i_{ml}}$ denotes an ML-based function[5]. The general formulation combines the ML model in Equation (1) and the analytical model in Equation (2). Now we start to introduce each part and explain its advantages.

First, we design an analytical resource function $\boldsymbol{F^i_{res}}(C_i)$ 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.

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

Second, we propose the ML model $\boldsymbol{F^i_{ml}}(C_i, E_i)$ 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 $\boldsymbol{F^i_{ml}}$ and resource function $\boldsymbol{F^i_{res}}$ are multiplied to generate the final power estimation. The finalized PANDA formulation is shown below.

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

As Equation (5) shows, PANDA adopts a simple multiplication to aggregate the ML model $\boldsymbol{F^i_{ml}}$ and analytical resource function $\boldsymbol{F^i_{res}}$. Although there may not be a definite answer, here we share three intuitive thoughts behind this decision. 1) As the I-Cache example in Equation 4 shows, our identified resource function for each component is theoretically roughly proportional to the power consumption. Therefore, there should be a linear relationship between $\boldsymbol{F^i_{res}}$ and power prediction, multiplication is obviously the simplest option. 2) A possible alternative is to directly incorporate the resource function $\boldsymbol{F^i_{res}}$ as an input feature of ML model $\boldsymbol{F^i_{ml}}$. But this is not a good option. Just like existing ML solutions calibrating McPAT, the ML model accuracy degrades significantly when training data is limited. When only used as ML model input, the strength of the analytical resource function is not fully utilized to tackle the data availability problem. 3) Another interesting perspective will be provided by the analysis in Section IV-C, it demonstrates that such multiplication in Equation (5) makes ML model learn to predict power/F_{res}, whose

[5]Please notice that the actual functions of $\boldsymbol{F^i_{agg}}$, $\boldsymbol{F^i_{res}}$, $\boldsymbol{F^i_{ml}}$ in PANDA are different from Equation (1)(2). We use the same notation to simplify the expression and demonstrate the general form of our formulation.

Component	$\boldsymbol{F^i_{res}}$	Component	$\boldsymbol{F^i_{res}}$
I-Cache	ICacheWay * ICacheFetchBytes	BP	FetchWidth
ISU	$f_{\text{ReserveStationNum}}$(DecodeWidth)	IFU	DecodeWidth
Regfile	IntPhyRegister + FpPhyRegister	I-TLB	ICacheTLBEntry + bias
D-Cache	DCacheWay * MemIssueWidth	RNU	DecodeWidth
LSU	LDQEntry + STQEntry	ROB	RobEntry
D-TLB	DCacheTLBEntry + bias	FU Pool	1
Other Logic	DecodeWidth + bias		

TABLE II: PANDA's resource function $\boldsymbol{F^i_{res}}$ of each major component i in the target out-of-order CPU core. They are derived based on the background knowledge of CPU architecture design.

distribution across different configurations is much more uniform than the power value alone. The result implies that ML model should learn power/F_{res} better when the training data is limited. Combining these three reasons, we believe multiplication is the correct option, and it indeed provides excellent performance.

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

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

D. Resource Functions and ML Model of PANDA

Table II shows our proposed key resource functions[6] $\boldsymbol{F^i_{res}}$ for all major components in the target out-of-order CPU core. We introduce how we derive these functions below.

Similar to the aforementioned I-Cache example, for an N-way set-associative L1 data cache (D-Cache), the power is roughly proportional to the number of cache ways (i.e., *DCacheWay*). In addition, modern CPUs support serving multiple read requests simultaneously to improve throughput [15]. The power consumption is proportional to the number of simultaneously accessed cache lines. This simultaneous cache access number typically equals the number of memory-access instructions issued each time (i.e., *MemIssueWidth*). Therefore, we propose $\boldsymbol{F^i_{res}}$ = *DCacheWay* * *MemIssueWidth*.

❶ **Frontend.** The Frontend part of the Out-of-Order CPU consists of 4 main components, including BP, IFU, I-TLB, and I-Cache. The I-Cache has been discussed. (1) The branch predictor (BP) is one of the most crucial parts of the frontend of modern OoO CPUs. It has a significant impact on CPU performance. A more accurate BP requires larger SRAM-based or register-based tables like the branch history table, leading to higher power consumption. In many modern CPUs, the size of the branch predictor is designed to scale proportionally with the number of instructions being fetched each time (i.e., *FetchWidth*) at the frontend. Therefore, we set $\boldsymbol{F^i_{res}}$ = *FetchWidth* as the BP's resource function. (2) For Instruction Fetch Unit (IFU), the size of its instruction fetch buffer and decode logic depends on the *DecodeWidth*. In comparison, the impact of other components on the IFU is secondary. Therefore, we set $\boldsymbol{F^i_{res}}$ = *DecodeWidth* for IFU. (3) 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 $\boldsymbol{F^i_{res}}$ = *ICacheTLBEntry* + bias for I-TLB, with the bias denoting a constant power value. Note that the bias will be readily estimated by fitting this linear function on training data, as long as there are more than one training samples.

❷ **Execution.** The Execution part of the Out-of-Order CPU consists of 5 main components, including RNU, ROB, ISU, Regfile, and FU Pool. (1) For the renaming unit (RNU), the RNU typically has a renaming width equal to the *DecodeWidth*. Therefore, we set $\boldsymbol{F^i_{res}}$ = *DecodeWidth* for RNU. (2) The reorder buffer (ROB), the power consumption mainly depends on the size of it, so the power consumption is proportional to the number of rob entries (i.e., *RobEntry*). Therefore, we set $\boldsymbol{F^i_{res}}$ = *RobEntry* for ROB. (3) The issue unit (ISU) is a critical part of OoO CPUs as it manages instruction scheduling and pipeline information. The reserve stations, which store the scheduling information, are the main component of ISU and largely affect the ISU power consumption. The number of reserve station entries typically depends on the number of instructions being decoded each time (i.e., DecodeWidth). We set $\boldsymbol{F^i_{res}}$ = $f_{\text{ReserveStationNum}}$(*DecodeWidth*), where the $f_{\text{ReserveStationNum}}$ is a look-up table that maps *DecodeWidth* to the number of reserve stations. It will be available as part of the CPU design. (4) The Regfile is mainly comprised of integer physical registers and float physical registers. So the power is proportional to the sum of the integer physical register size and the float physical register size. Therefore, we set $\boldsymbol{F^i_{res}}$ = *IntPhyRegister* + *FpPhyRegister* for Regfile. (5) The FU Pool is very complex, comprised of a variety of function units, and each function unit has different power characteristics. Therefore, we set $\boldsymbol{F^i_{res}}$ = 1 for FU Pool, handing this over to the ML function.

❸ **Memory Access.** The Memory Access part of the Out-of-Order CPU consists of 3 main components, including LSU, D-TLB, and D-Cache, the D-Cache has been discussed. (1) The load store unit (LSU) is mainly comprised of the load queue and store queue. Its power consumption is closely tied to the total number of entries in these queues, denoted as *LDQEntry* + *STQEntry*. Therefore, we set $\boldsymbol{F^i_{res}}$ = *LDQEntry* + *STQEntry* for LSU. (2) For the D-TLB, this component is very similar to I-TLB. Similarly, we set $\boldsymbol{F^i_{res}}$ = *DCacheTLBEntry* + bias for D-TLB.

❹ **Other Logic.** 'Other Logic' is the most complex part of the CPU, consisting of all other control and pipeline logic except existing components. It may seem that the power consumption of 'Other Logic' is difficult to estimate, but we have identified *DecodeWidth* as a useful indicator. *DecodeWidth* determines the number of instructions that can be decoded simultaneously, and it typically also equals the width of several subsequent pipeline stages, such as the integer rename width, ROB width, and commit width, etc. Therefore, *DecodeWidth* can be referred to as the general *pipeline width* of the whole CPU design. A large portion of the control logic and pipeline logic scale with this pipeline width, while other parts remain unchanged even as other components of the CPU scale out. Therefore, we derive the resource function as $\boldsymbol{F^i_{res}}$ = *DecodeWidth* + bias, with the bias denoting a constant power value. Similar to I-TLB, the bias can be estimated using training data.

As for the ML model $\boldsymbol{F^i_{ml}}$ of each component, we all adopt Gradient Boosting Trees [16] like XGBoost [17], a widely-adopted machine learning algorithm for tabular data type, to build regressors. To avoid introducing our engineers' bias during ML model hyper-parameter tuning, for all these XGBoost ML models in PANDA, we simply adopt the default hyper-parameters (i.e., max depth=6, num of estimatros=100) without any further parameter tuning. PANDA is already sufficiently accurate in this case.

E. Other Design Quality Prediction

Besides power prediction, PANDA also supports evaluation of other design qualities including area, performance (i.e., the number of cycles to complete a given workload), and energy. (1) The area model is similar to aforementioned component-level power model, but only uses configuration parameters C_i as features, without including workload event parameters E_i. (2) For performance prediction, we observe that gem5 [14] achieves a relatively accurate correlation R, but with obvious absolute errors. Therefore, we simply develop one overall performance model to calibrate gem5 [14]. Specially, we directly use the ratio between ground-truth execution cycle numbers and the gem5 evaluation as the training label. The input features include all configuration parameters C and selected event parameters E. Considering the branch prediction and long latency of memory access are critical for the performance, we select these key event parameters from E as features: {numCycles, idleCycles, branchPred condPredicted, branchPred condIncorrect, icache overallMisses, icache ReadReq.-mshrMisses, dcache ReadReq.misses, dcache WriteReq.misses, dcache overallMisses, dcache overallMshrMisses}. (3) Finally, PANDA can naturally evaluate the energy consumption of a given workload, by combining its performance prediction with power prediction.

[6]In the remaining of this paper, we may directly denote the resource function $\boldsymbol{F^i_{res}}(C_i)$ as $\boldsymbol{F^i_{res}}$ for simplicity.

Configuration Parameter	C1	C2	C3	C4	C5	C6	C7	C8	C9	C10	C11	C12	C13	C14	C15	SP1	SP2
FetchWidth	4	4	4	4	4	8	8	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	1	5
FetchBufferEntry	5	8	16	8	16	24	18	24	30	24	32	40	30	35	40	10	40
RobEntry	16	32	48	64	64	80	81	96	114	112	128	136	125	130	140	16	140
IntPhyRegister	36	53	68	64	80	88	88	110	112	108	128	136	108	128	140	36	140
FpPhyRegister	36	48	56	56	64	72	88	96	112	108	128	136	108	128	140	36	140
LDQ/STQEntry	4	8	16	12	16	20	16	24	32	24	32	36	24	32	36	4	36
BranchCount	6	8	10	10	12	14	14	16	16	18	20	20	18	20	20	6	20
MemIssue/FpIssueWidth	1	1	1	1	1	1	1	1	2	1	2	2	2	2	2	1	2
IntIssueWidth	1	1	1	1	2	2	2	3	3	4	4	4	5	5	5	1	5
DCache/ICacheWay	2	4	8	4	4	8	8	8	8	8	8	8	8	8	8	2	2
DTLBEntry	8	8	16	8	8	16	16	16	32	32	32	32	32	32	32	8	32
DCacheMSHR	2	2	4	2	2	4	4	4	4	4	4	8	8	8	8	2	8
ICacheFetchBytes	2	2	2	2	2	4	4	4	4	4	4	4	4	4	4	4	4

TABLE III: The configurations that we used in our experiment. The first 15 configurations (named C1-C15) are normal configurations, which is divided into 5 domains depending on the DecodeWidth, the last two configuration (named SP1, SP2) are special case that will be used in the case study part. To be specific, the SP1 is derived from C1, which is a CPU with a large BP but small other components, the SP2 is derived from C15, which is a CPU with a small D-Cache and I-Cache but large other components.

F. Transferring to New Technology

As the process technology node keeps shrinking, when power model is trained on labels from a certain technology node, it is often desirable to know the power of an unknown design in a new technology node. This task is obviously challenging. Designers often simply scale the power consumption from known technology to a new technology based on $P = CV^2$, but it is over-simplified. We propose a new ML method to predict the power consumption of an unknown configuration under a new unknown technology node. It naturally involves two steps. 1) Predict power of the unknown configuration with PANDA. The result corresponds to the technology node where PANDA is trained. We refer to it as the source node. 2) Based on the result, transfer the prediction to the new unknown technology node, named the target node. The insight to achieve the transferring is, when transferring to a new technology node, the impact on large designs i.e. BOOM CPU, and those very small designs is similar. It gives us an opportunity to predict the transformation of large designs using the ground-truth of small designs, which can be generated without too much cost.

We build an ML model for this transferring task. For each design, we use the ratio between power under target technology and source technology as the label. The features include (1) PANDA predicted power consumption under source node, (2) the ratio between target and source's scale and voltage. For example, when the source node is 28nm 0.8V, the target one is 40nm 1.1V, the feature is $40/28$ and $1.1/0.8$, (3) the directly-scaled power using $P = CV^2$. Take the previous 28nm to 40nm example, this scaled power is PANDA-predicted power at 28nm multiplied by $40/28 * (1.1/0.8)^2$. We collect ground-truth from small designs under multiple different technology nodes and train this transferring model only using small designs. Please notice that only one model is trained to perform transferring among multiple technology nodes. After training, this model can make transfer predictions between any two technology nodes.

III. Experimental Results

A. Experiment Setup

To evaluate our method, we generate a dataset by collecting RTL code and performing RTL simulation, with Chipyard [18] v1.8.1. For a fair comparison with prior works [9], we employed 15 similar RISC-V BOOM [15] CPU configurations in Table III, named C1 to C15, ranging from small to large design sizes. Similar to prior works [10], [13], we further divided these 15 configurations into five domains based on their *DecodeWidth*, which is a key configuration parameter that affects multiple pipeline stages. The detailed configurations are given in Table III. For vector-based power simulation, we used eight workloads in the riscv-tests [19] suite, including dhrystone, median, multiply, qsort, rsort, towers, spmv, and vvadd.

We performed RTL simulation at 1GHz with Synopsys VCS® [20]. The logic synthesis and ground-truth power simulation are performed with Synopsys Design Compiler® [21] and PrimePower [2], respectively. For the technology node, we used the TSMC 40nm standard cell library and the corresponding Memory Compiler. Furthermore, in our evaluation of cross-technology node prediction, we also adopt the TSMC 28nm and 65nm standard cell libraries.

B. Summary of Baseline Methods

We compared PANDA with representative prior works, including (a) McPAT (+gem5 [14] [22]) [3], a typical analytical model, (b) PowerTrain [11], a widely-adopted lightweight ML-based model, and (c) McPAT-Calib [9], which is the state-of-the-art ML-based model. All of these three works rely on McPAT as part of the power model. Additionally, (d) TCAD'17 [23], a representative on-chip power meter design method based on performance counters, is included as a baseline. On-chip power meters are not originally designed for architecture-level power modeling, we include them for the completeness of our experiment. For this task, the performance counters are replaced with the event parameters generated by gem5 [14].

For a recent work ASPDAC'23 [10], it mainly proposed a transfer learning algorithm to predict the power of configurations in the unknown domain. But we emphasize that such transfer learning still requires sampling in the unknown domain, which necessitates the RTL implementation and subsequent synthesis of the sampled configurations. It is quite costly, since although the Chipyard can generate RTL code automatically, it is not a always the case in the industry, where implementing RTL for a new CPU configuration may require significant engineering effort. What's more, the core power prediction part of ASPDAC'23 is similar to McPAT-Calib [9], by adopting multi-layer perceptron as its ML model. Therefore, we do not present it separately.

Besides using prior works as baselines, we also include two extra baselines. (e) To further demonstrate the potential of McPAT without inaccuracies introduced by its internal coefficients, an ideal scaling factor is derived based on ground-truth power of all configurations. It scales the power prediction of McPAT towards the ground-truth. This baseline with superior accuracy than McPAT is named McPAT-plus. (f) We also build a much weaker variant of PANDA with limited architecture-level knowledge, named the Component-level model, as shown in Fig. 3(c). It builds ML models for each component with related configuration parameters and event parameters in Table I as features. But it does not adopt the resource functions in PANDA.

We conduct multiple experiments with different amount of training data. When the number of known configurations as training data is n

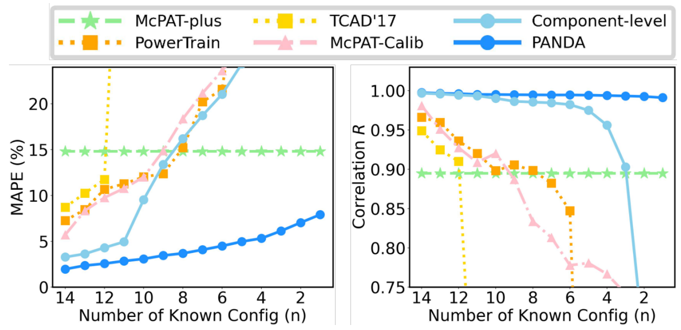


Fig. 4: The MAPE and R of different models under different number of unknown configurations. The McPAT-plus is a correctly calibrated version of McPAT, directly scaled with ground-truth. PANDA's superior accuracy over baselines increases as the training data decrease.

($n \in [1, 14]$), the $15 - n$ unknown configurations will be testing data[7]. The experiment setup and actual data used for training and testing are strictly the same for all methods during comparison. We evaluate performance with the mean absolute percentage error (MAPE)[8] and correlation coefficient R averaged over all testing configurations.

C. Power Prediction Results

Fig. 4 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 [9] and PowerTrain [11] in Fig. 4. By integrating architecture-level knowledge into its resource function, PANDA can efficiently capture how the configuration parameters modification will affect the power based on very limited training data. 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 [23] naturally performs poorly since it is designed for on-chip power meter development instead of this task.

As for analytical model baseline McPAT-plus, as a correctly-scaled version of McPAT, its accuracy remains unchanged in Fig. 4 regardless of training data amount. Without ML model, its overall accuracy is limited. By unifying both analytical and ML techniques, PANDA outperforms it significantly even when there is only one known configuration ($n = 1$)[9] 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.

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. By training ML models at 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 with architecture knowledge.

In Fig. 5, 7, 8, we further visualize the detailed prediction results when the number of known configurations (n) for training is 14, 5, and 1, respectively. Additionally, in Fig. 6 we introduce a new scenario called the 'unknown domain', where each time there are 4 domains as known training sets (with 12 configurations) and 1 domain as the testing set (with 3 configurations). The testing set will traverse all 5 domains to generate predictions on all 15 configurations.

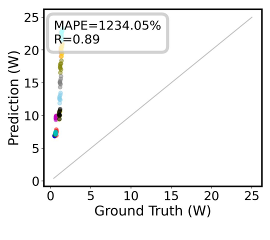
D. Prediction of Other Design Qualities

Besides power prediction, Table IV shows PANDA's prediction accuracy on area, performance, and energy. Since the majority of prior power models do not further evaluate these design qualities, in this part, we use McPAT as the baseline for area prediction, and gem5 as the baseline for performance prediction. For the energy baseline, we collect the performance predicted by gem5 and power predicted by McPAT-Calib to compute the energy. Table IV also adopts the 'unknown-domain' scenario, where each time 1 unknown domain is

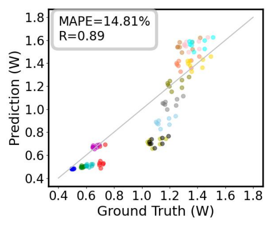
[7]For example, when n is 13, there are 13 known configurations for training and 2 testing configurations for testing. We sequentially traverse testing sets with neighboring configurations (C1,C2), ..., (C14,C15), and (C15,C1). Altogether 15 models are trained & evaluated, corresponding to each testing set. These predictions on each testing configuration are averaged as the final prediction.

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

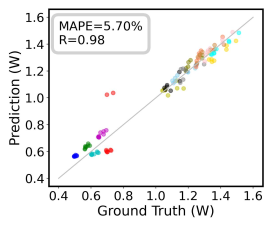
[9]For $n = 1$, as a special case, the bias in a few resource function has to be derived based on designers observation. But it is strictly fitted with training data when $n \geq 2$.



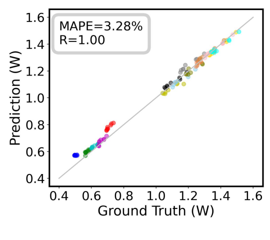
(a) McPAT



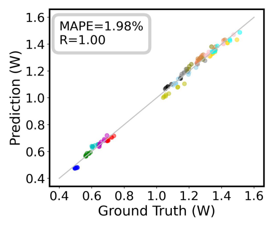
(b) McPAT plus



(c) McPAT-Calib

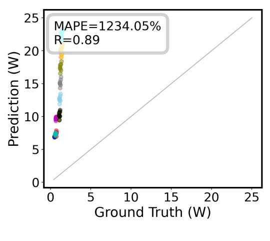


(d) Component-level Model

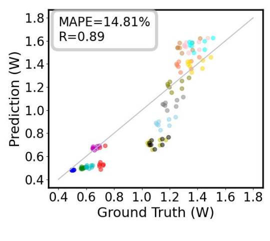


(e) PANDA

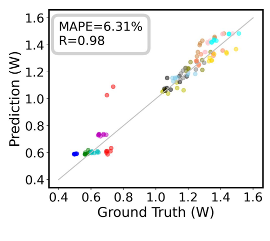
Fig. 5: Accuracy comparison for the known 14 (unknown 1) configuration scenario.



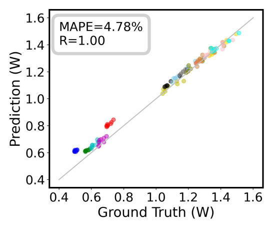
(a) McPAT



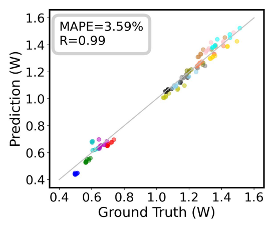
(b) McPAT plus



(c) McPAT-Calib

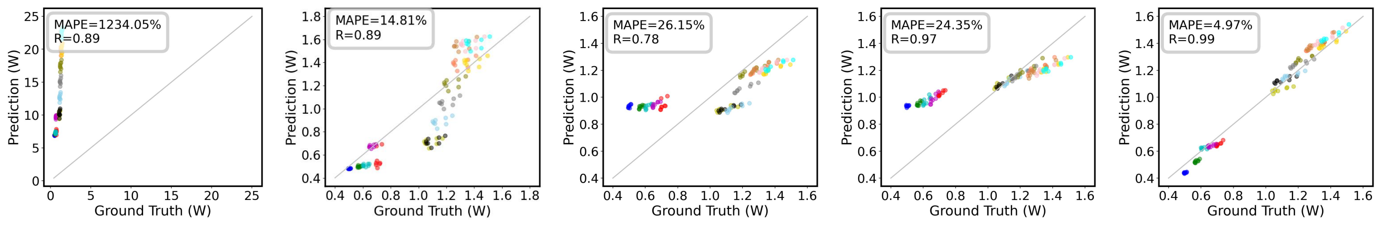


(d) Component-level Model



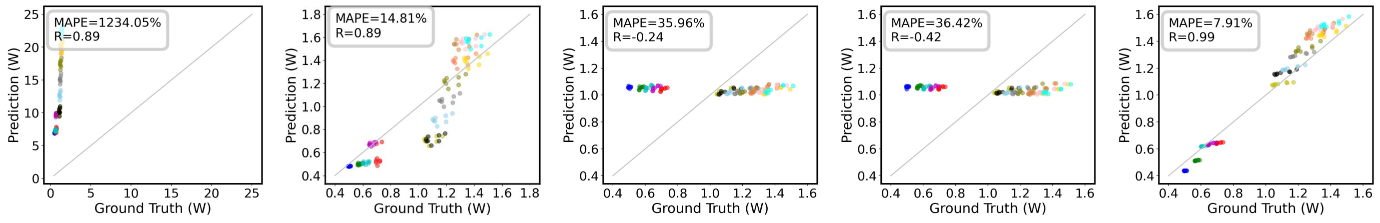
(e) PANDA

Fig. 6: Accuracy comparison for the unknown-design-domain scenario.



(a) McPAT (b) McPAT-plus (c) McPAT-Calib (d) Component-level Model (e) PANDA

Fig. 7: Accuracy comparison for the known 5 (unknown 10) configuration scenario.



(a) McPAT (b) McPAT plus (c) McPAT-Calib (d) Component-level Model (e) PANDA

Fig. 8: Accuracy comparison for the known 1 (unknown 14) configuration scenario.

Design Quality	Baseline			PANDA	
	Baseline Method	MAPE(%)	R	MAPE(%)	R
Area	McPAT	416.56	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 IV: The comparison of Area, Performance, and Energy prediction between baseline and PANDA

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

used as the testing set. The baseline methods achieve good correlation in all 15 configurations but with a huge absolute error value. In comparison, PANDA achieves even higher correlation and keeps the error within 10%.

E. Cross-Technology Prediction

We evaluate the cross-technology prediction accuracy of PANDA with three different technology nodes, TSMC 28nm 0.8V, TSMC 40nm 1.1V, and TSMC 65nm 1.2V. Our proposed transfer model is trained based on about 20 small designs implemented with all three technologies. On average, these small designs consist of only thousands of gates, in contrast with 0.3 million gates in the smallest BOOM configuration C1. For any pair of source and target technology nodes, the transfer model will predict the power of an unknown design configuration at the target technology node, based on PANDA power model's prediction at the source node.

We calculate 1) the original prediction of source technology based on PANDA's power model; 2) the directly-scaled prediction towards the target technology based on CV^2; 3) the transferred prediction with our proposed transferring model. We compare these predictions with ground-truth labels at the target technology node, evaluating the MAPE. Their accuracies are shown in Table V. The average MAPE of scaled prediction and model prediction are 22.16% and 10.40%, respectively, showing that PANDA outperforms analytically scaling.

IV. Discussion

To further demonstrate the advantages of PANDA, we present two application scenarios as case studies. Finally, we present an analysis to verify the correctness of the design of PANDA.

A. Case Study 1: Power Prediction for Special Configurations

For existing 15 configurations in the experiment, from the smallest C1 to the largest C15, all component parameters monotonically increase with a similar trend. As a result, although the total power increases from C1 to C15, the power percentage of each component does not vary much. However, in practice, this is not always the case, as realistic configurations may have different percentages of power consumption for some key components. To study this scenario, we design two special cases SP1 and SP2, as already shown in Table III. SP1 is with a large BP but small other components, while SP2 is with a small D-Cache and I-Cache but large other components. For such special-case designs, prior ML methods may perform poorly due to a lack of awareness of CPU hierarchy, which makes it difficult to capture how each component contributes to the total power consumption. In comparison, PANDA can well handle any configuration parameter combinations by modeling each component separately.

Fig. 9 compares the predictions on SP1 and SP2 by McPAT-Calib and PANDA trained with C1 to C15, showing the MAPE[10]. Despite all 15 configurations are used for training, McPAT-Calib is very inaccurate with MAPE=22.8%. Compared with McPAT-Calib's average MAPE=5.7% when trained with 14 known configurations in Fig. 4, this significantly lower accuracy indicates the challenges of special cases. In comparison, PANDA prediction remains accurate with MAPE<5% for these special cases. It implies that PANDA can maintain a reasonably high accuracy on almost any new configuration.

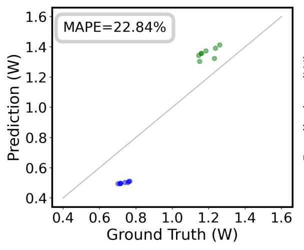
B. Case Study 2: Design Space Exploration with PANDA

Design space exploration (DSE) is an important task of CPU architects. For example, given a power constraint, architects explore which configuration achieves the highest performance. However, generating ground-truth power and performance of each design configuration

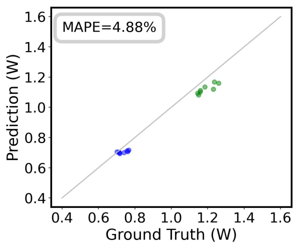
Design Name	Config Parameter (same order as TABLE III)	Power	Performance
Engineer Design	4, 3, 24, 96, 96, 96, 24, 16, 2, 4, 2, 8, 8, 2	0.79	2.08
DSE Design	4, 4, 32, 128, 96, 96, 32, 16, 2, 4, 2, 8, 4, 2	0.80	2.30

TABLE VI: The comparison of design selected by engineer and design selected by DSE, the parameters are in the same order as TABLE III. Power unit is (W) and performance is normalized by C1's performance.

[10]Since there are only two configurations, correlation R is not a good metric.



(a) McPAT-Calib



(b) PANDA

Fig. 9: The prediction of two special cases with different patterns in configuration parameters from the training set (C1-C15), blue means a CPU with a large Frontend but small other components, green means a CPU with a small I-Cache and D-Cache but large other components.

requires timing-consuming design implementation. PANDA, as an accurate architecture-level power and performance model, naturally supports efficient DSE.

We assume a DSE task where configurations C1-C15 are known and PANDA is already trained by them. Since none of these configurations has the power value in the range of 0.7 to 1W, we let PANDA explore this unknown region. The specific task is to select a configuration with maximum performance, with power consumption less or equal to 0.8W. Here we compare the performance of PANDA and human engineers, who manually set configuration parameters using the ground-truth power of C1-C15 as a reference.

Then we start DSE by efficiently predicting almost all reasonable configuration parameter combinations in the design space with PANDA, then select high-performance configurations that satisfy the power constraint. We tolerate configurations with predicted power slightly higher than the constraint. This process is fast due to the efficiency of PANDA. We then start to implement selected configurations based on the rank of the predicted performance values. This process stops when one configuration's implementation proves to satisfy the power constraint. Due to the accuracy of PANDA, we reach the power constraint after very few trials. Human engineers go through a similar validation process, they craft and implement new configurations until power constraint is met.

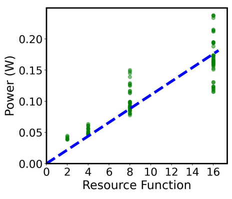
The ground-truth power and performance of PANDA-explored configuration are shown in the Table VI, with a power of 0.80 W and a performance of 2.30. Compared with the engineer's selection, which is also close to the power constraint, it achieves 0.22 higher performance. It demonstrates PANDA's potential in DSE applications. Compared with existing DSE works that invoke VLSI design flow iteratively, once PANDA is trained with a few known samples, it requires no more sampling or update for the space exploration task.

C. Resource Function and Power Analysis

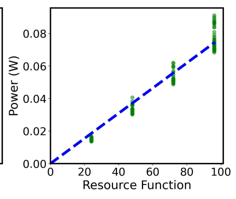
Finally, to verify the correctness of PANDA, we first visualize the correlation between ground-truth component power and the corresponding resource function $\boldsymbol{F^i_{res}}$ in Fig. 10. For D-Cache and ISU, power mainly scales proportionally with the resource function (x-axis). As for the Other Logic, we show *DecodeWidth* in x-axis, and its power is also proportional to the resource function *DecodeWidth* + bias. These data patterns indicate the correctness of PANDA's resource function $\boldsymbol{F^i_{res}}$, which will be multiplied by ML model to generate power prediction, as defined by Equation (5).

From another perspective, Fig. 10 also shows that when the resource function is fixed 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 $\boldsymbol{F^i_{ml}}$. According to Equation (5), the ML model $\boldsymbol{F^i_{ml}}$ is actually trained to predict the power divided by the resource function (i.e., power/$\boldsymbol{F^i_{res}}$).

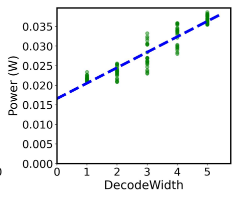
To analyze such vertical power variation for different resource function values, we visualize the power distribution of D-Cache in



(a) D-Cache

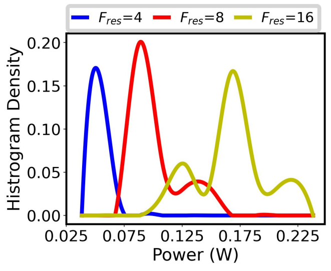


(b) ISU

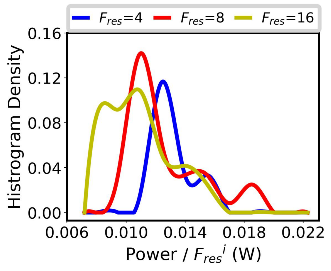


(c) Other Logic

Fig. 10: Component power vs its resource function $\boldsymbol{F^i_{res}}$, except (c), which demonstrates the relationship between power and DecodeWidth, because the bias in its resource function needs to be estimated.



(a) Original Distribution



(b) Distribution for $\boldsymbol{F^i_{ml}}$ to learn

Fig. 11: Distribution of D-Cache power, it corresponds to the power distribution of points of Fig. 10(a). (a) Original power, learned by existing ML methods. (b) Power divided by resource function (power/$\boldsymbol{F^i_{res}}$), learned by PANDA's ML model. PANDA's ML part $\boldsymbol{F^i_{ml}}$ learns more similar distributions, benefiting accuracy when training data is limited.

Fig. 11(a). It corresponds to vertical points in Fig. 10(a)[11]. We further visualize the distribution of power/$\boldsymbol{F^i_{res}}$ in Fig. 11(b). The comparison between Fig. 11(a) and Fig. 11(b) shows an interesting pattern and provides another explanation of the superior performance of PANDA.

In Fig. 11(a), 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, then perform bad on testing designs from unknown other distributions. The gap between different distributions is large, causing an obvious prediction error. In comparison, PANDA actually trains the ML model to predict power/$\boldsymbol{F^i_{res}}$, as shown in Fig. 11(b). This power/$\boldsymbol{F^i_{res}}$ objective provides obviously more similar distributions. 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.

V. Conclusion

In this work, we propose PANDA, an architecture-level power model that unifies analytical and machine learning techniques. PANDA develops its own simpler analytical function for each component based on architecture knowledge, leaving more complex patterns to be learned by the ML part. It significantly outperforms state-of-the-art solutions, and maintains a high accuracy even with the very limited number of known configurations for training. Such a data-friendly solution lowers the barrier to adopting ML techniques by design teams, and thus PANDA is a compelling addition to the architects' toolbox.

VI. Acknowledgement

This work is partially supported by the Hong Kong Research Grants Council (RGC) ECS Grant 26208723, Semiconductor Research Corporation (SRC) GRC-CADT 3103.001/3104.001, and ACCESS – AI Chip Center for Emerging Smart Systems, sponsored by InnoHK funding, Hong Kong SAR. Also, the authors thank the help from Prof Bei Yu and Chen Bai at the Chinese University of Hong Kong.

[11] Configurations with $F_{res} = 2$ only account for 6% among all configurations of D-Cache displayed in Fig. 10(a). Therefore, we discard this small part in Fig. 11, only showing $F_{res} = 4, 8, 16$.

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, J. H. Ahn, R. D. Strong, J. B. Brockman, D. M. Tullsen, and N. P. Jouppi, "McPAT: An integrated power, area, and timing modeling framework for multicore and manycore architectures," in *Proceedings of the 42nd annual ieee/acm international symposium on microarchitecture (MICRO)*, 2009, pp. 469–480.

[4] A. Tang, Y. Yang, C.-Y. Lee, and N. K. Jha, "Mcpat-pvt: Delay and power modeling framework for finfet processor architectures under pvt variations," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 23, no. 9, pp. 1616–1627, 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 Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 28, no. 10, pp. 2146–2156, 2020.

[6] D. Brooks, V. Tiwari, and M. Martonosi, "Wattch: A framework for architectural-level power analysis and optimizations," *ACM SIGARCH Computer Architecture News*, vol. 28, no. 2, pp. 83–94, 2000.

[7] S. L. Xi, H. Jacobson, P. Bose, G.-Y. Wei, and D. Brooks, "Quantifying sources of error in mcpat and potential impacts on architectural studies," in *2015 IEEE 21st International symposium on high performance computer architecture (HPCA)*. IEEE, 2015, pp. 577–589.

[8] T. Nowatzki, J. Menon, C.-H. Ho, and K. Sankaralingam, "Architectural simulators considered harmful," *IEEE Micro*, vol. 35, no. 6, pp. 4–12, 2015.

[9] J. Zhai, C. Bai, B. Zhu, Y. Cai, Q. Zhou, and B. Yu, "McPAT-Calib: A RISC-V BOOM microarchitecture power modeling framework," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, vol. 42, no. 1, pp. 243–256, 2022.

[10] J. Zhai, Y. Cai, and B. Yu, "Microarchitecture power modeling via artificial neural network and transfer learning," in *2023 28th Asia and South Pacific Design Automation Conference (ASPDAC)*, 2023.

[11] W. Lee, Y. Kim, J. H. Ryoo, D. Sunwoo, A. Gerstlauer, and L. K. John, "Powertrain: A learning-based calibration of mcpat power models," in *2015 IEEE/ACM International Symposium on Low Power Electronics and Design (ISLPED)*. IEEE, 2015, pp. 189–194.

[12] B. C. Lee and D. M. Brooks, "Illustrative design space studies with microarchitectural regression models," in *2007 IEEE 13th International Symposium on High Performance Computer Architecture*. IEEE, 2007, pp. 340–351.

[13] C. Bai, Q. Sun, J. Zhai, Y. Ma, B. Yu, and M. D. Wong, "BOOM-Explorer: RISC-V BOOM microarchitecture design space exploration framework," in *2021 IEEE/ACM International Conference On Computer Aided Design (ICCAD)*. IEEE, 2021, pp. 1–9.

[14] N. Binkert, B. Beckmann, G. Black, S. K. Reinhardt, A. Saidi, A. Basu, J. Hestness, D. R. Hower, T. Krishna, S. Sardashti *et al.*, "The gem5 simulator," *ACM SIGARCH computer architecture news*, vol. 39, no. 2, pp. 1–7, 2011.

[15] J. Zhao, B. Korpan, A. Gonzalez, and K. Asanovic, "Sonicboom: The 3rd generation berkeley out-of-order machine," in *Fourth Workshop on Computer Architecture Research with RISC-V*, vol. 5, 2020.

[16] J. H. Friedman, "Greedy function approximation: a gradient boosting machine," *Annals of statistics*, pp. 1189–1232, 2001.

[17] T. Chen and C. Guestrin, "Xgboost: A scalable tree boosting system," in *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*, 2016, pp. 785–794.

[18] A. Amid, D. Biancolin, A. Gonzalez, D. Grubb, S. Karandikar, H. Liew, A. Magyar, H. Mao, A. Ou, N. Pemberton *et al.*, "Chipyard: Integrated design, simulation, and implementation framework for custom socs," *IEEE Micro*, vol. 40, no. 4, pp. 10–21, 2020.

[19] *RISC-V Tests*, https://github.com/riscv-software-src/riscv-tests, 2022.

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

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

[22] A. Roelke and M. R. Stan, "Risc5: Implementing the risc-v isa in gem5," in *First Workshop on Computer Architecture Research with RISC-V (CARRV)*, vol. 7, no. 17, 2017.

[23] M. J. Walker, S. Diestelhorst, A. Hansson, A. K. Das, S. Yang, B. M. Al-Hashimi, and G. V. Merrett, "Accurate and stable run-time power modeling for mobile and embedded cpus," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, vol. 36, no. 1, pp. 106–119, 2016.