FirePower: Towards a Foundation with Generalizable Knowledge for Architecture-Level Power Modeling

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

ABSTRACT

Power efficiency is a critical design objective in modern processor design. A high-fidelity architecture-level power modeling method is greatly needed by CPU architects for guiding early optimizations. However, traditional architecture-level power models can not meet the accuracy requirement, largely due to the discrepancy between the power model and actual design implementation. While some machine learning (ML)-based architecture-level power modeling methods have been proposed in recent years, the data-hungry ML model training process requires sufficient similar known designs, which are unrealistic in many development scenarios.

This work proposes a new power modeling solution FirePower that targets few-shot learning scenario for new target architectures. FirePower proposes multiple new policies to utilize cross-architecture knowledge. First, it develops power models at component level, and components are defined in a power-friendly manner. Second, it supports different generalization strategies for models of different components. Third, it formulates generalizable and architecture-specific design knowledge into two separate models. FirePower also supports the evaluation of the generalization quality. In our experiments, FirePower can achieve a low error percentage of 5.8% and a high correlation R of 0.98 on average only using two configurations of target architecture. This is 8.8% lower in error percentage and 0.03 higher in R compared with directly training McPAT-Calib baseline on configurations of target architecture.

CCS CONCEPTS

• **Hardware → Power estimation and optimization**.

KEYWORDS

Power model, machine learning

ACM Reference Format:
Qijun Zhang, Mengming Li, Yao Lu, Zhiyao Xie. 2025. FirePower: Towards a Foundation with Generalizable Knowledge for Architecture-Level Power Modeling. In *30th Asia and South Pacific Design Automation Conference (ASPDAC '25), January 20–23, 2025, Tokyo, Japan.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3658617.3697554

*Corresponding Author

ASPDAC '25, January 20–23, 2025, Tokyo, Japan

ACM ISBN 979-8-4007-0635-6/25/01
https://doi.org/10.1145/3658617.3697554

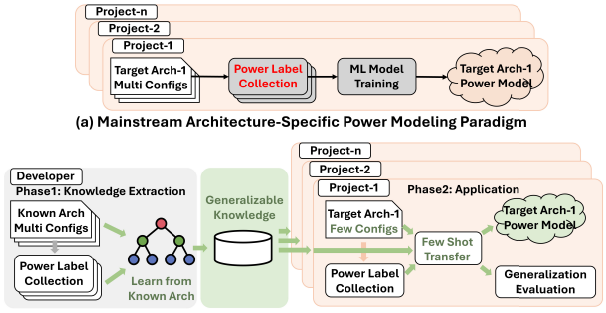


Figure 1: Proposed power modeling paradigm FirePower vs. existing architecture-specific paradigm. FirePower targets few-shot learning for new target architectures. It extracts general knowledge from an already known architecture, providing a "foundation" to support modeling new architectures.

1 INTRODUCTION

Power efficiency is a critical design objective for processor design. With the increasing design complexity, it takes significant time and effort in power optimization. As a result, a fast and highly accurate architecture-level power modeling method is greatly needed for early power evaluation prior to RTL implementation. Traditional analytical architecture-level power models such as McPAT [19] and Wattch [9] are often inaccurate, largely due to the discrepancy among architecture-level simulator, power model, and real target CPU. This has been discussed in many prior works [26, 30]. Despite some works [13, 25] updating internal design of analytical power models, they require significant human efforts and are primarily developed in-house to cater to proprietary designs.

In recent years, ML-based architecture-level power modeling methods [18, 30–32] have been explored and demonstrated better accuracy by calibrating analytical models with ML models. However, as Fig. 1(a) shows, most ML-based power models are developed for a specific architecture, and are only applicable to new configurations under exactly the same architecture. For example, most prior works train and test power models on BOOM CPUs [33] only. Training and testing are performed on different BOOM configurations, sharing obvious similarities. Building an ML-based power model requires sufficient ground-truth power labels of known configurations of target architecture [18, 30, 31]. For a new project developing a slightly different architecture, the whole power model must be retrained from scratch based on a brand-new training dataset.

In practice, for an ongoing project targeting a specific architecture, there are not many already accomplished designs available to provide training labels. If collecting labels from scratch, the process can be highly expensive: For each design configuration, its label collection requires register-transfer level (RTL) implementation, synthesis, and simulation with workloads. RTL implementation can be especially tedious. A more practical scenario is, only a few

design configurations under the target architecture are available to provide training labels. This is a typical few-shot learning scenario. The existing architecture-specific power modeling paradigm suffers from limited accuracy when only a few labels are available.

Motivated by the limitation of architecture-specific methods, we propose to extract general knowledge that can be applied across architectures. This is possible since different architectures still have similarities. Take out-of-order CPUs as an example, despite different design decisions, general design principles are similar, leading to partially similar power characteristics. However, separating general and specific knowledge at the architecture level is challenging since there is no clear standard of general or architecture-specific part of power. To the best of our knowledge, no prior ML power models have explicitly explored this topic at the architecture level.

In this work, we propose a two-phase power modeling paradigm named FirePower, as shown in Fig. 1(b). (1) The first phase is named knowledge extraction. This first phase can be time-consuming, but it is performed only once and by us, FirePower developers. Developers will first collect data on one known architecture (e.g., BOOM CPU series). Sufficient power labels will be collected for multiple configurations with good coverage for the design space. The framework will extract generalizable knowledge from this architecture. This general knowledge will provide a *foundation* for the few-shot power modeling tasks on other architectures. (2) The second phase will utilize the foundation with general knowledge to build the power model for each new target architecture. With only a few available configurations of the target architecture (i.e., few-shot), the power model significantly outperforms existing architecture-specific models, which have to be trained from scratch.

The contributions of this work can be summarized below.

- We analyze the limitation of the architecture-specific power modeling paradigm when applied to different target architectures. Then we propose a new paradigm FirePower[1] that targets the few-shot learning scenario for new target architectures by different users.
- To the best of our knowledge, FirePower is the first data-driven architecture-level power model that explores cross-architecture design knowledge. This exploration contributes to the understanding of the gap among architectures. This framework is fully automated, without requiring additional designer knowledge about any target architecture.
- FirePower proposes multiple new policies to use cross-architecture knowledge. 1) It develops models at component level, and components are defined in power-friendly manner. 2) It supports different generalization strategies for different components. 3) It formulates generalizable and architecture-specific design knowledge into two separate models.
- To detect the risk of huge differences between known and target architectures, FirePower supports the evaluation of the generalization quality for any target architecture. This helps provide the applicable scope of the method.
- We evaluate FirePower using two widely adopted open-source RISC-V CPU designs: BOOM and XiangShan. It demonstrates that FirePower can achieve a low MAPE of 5.8% and a high correlation R of 0.98 on average only using two configurations of the target architecture. It achieves 8.8% lower MAPE and 0.03 higher R compared with directly training McPAT-Calib on configurations of the target architecture.

[1]It is open-sourced at https://github.com/hkust-zhiyao/FirePower

2 RELATED WORK

Standard power estimation flow includes RTL implementation, logic synthesis, RTL simulation, and power simulation [23, 24]. In recent years, design-specific ML power models have been proposed [11, 12, 14–16, 20, 22, 27, 28, 34] for RTL stage using RTL signals as input. However, they still require RTL implementation, and a new model needs to be developed from scratch for each design. For pre-RTL stage, an accurate architecture-level power model is greatly needed.

Architecture-level power modeling takes architecture-level hardware parameters (denoted as H) and event statistics (denoted as E) as input to calculate power. Hardware parameters are the parameters used to describe the CPU configuration, such as $FetchWidth$ and $DCacheWay$. Event statistics are the information generated by running the workloads on the architecture-level performance simulator, such as the number of cache hits and branch instructions.

Traditional analytical architecture-level power models like McPAT [19] calculate energy consumption for each event based on hardware parameters, then divide the accumulation of them by the execution time to calculate power. Wattch [9] also adopts a similar method. It calculates the power of each cycle by accumulating the energy consumption for each event in this cycle and then dividing it by the time of a cycle. However, because of the discrepancy among the architecture-level performance simulator, the power model, and the actual CPU design, these analytical models are often inaccurate.

To deal with the inaccuracy of the analytical power model, data-driven ML power models have been proposed in recent years [30–32]. One of the representative data-driven architecture-level power models is McPAT-Calib [30]. McPAT-Calib trains an ML model to calibrate McPAT output towards power labels. Denoting the output of McPAT as M, the McPAT-Calib can be formulated below.

$$P = \boldsymbol{F_{ml}}(H, E, M)$$

This ML model tries to capture the mapping from these features to the power, such mapping is obviously different for different architectures. In this case, it needs to be retrained from scratch when applied to a new architecture. The work of [31] uses the same formulation but performs transfer learning to a new domain of the same architecture design, where a domain means configurations with the same $DecodeWidth$. Therefore, it [31] still doesn't support cross-architecture power modeling. The PANDA [32] proposes human-crafted resource functions to achieve few-shot learning where the known configuration is limited. However, designing the resource function requires significant engineering expertise for each target architecture. This is not an automated solution.

3 PROBLEM FORMULATION

Here we introduce problem formulation. FirePower developer starts with a known architecture. The design space of the known architecture is already well-explored by the developer so there are sufficient known configurations. The power simulation of these configurations has also been performed with multiple workloads. The collection of the dataset is denoted as $\mathcal{D}_{known}$.

When applying FirePower, there are multiple in-progress projects that need to build power models for their own target architectures at a low cost. For these ongoing design projects, there are only a few available configurations of target architectures. The dataset with a

few configurations of target architecture and corresponding power labels is denoted as $\mathcal{D}_{target}$. In experiments, we test three scenarios when there are 2, 3, or 4 available configurations in $\mathcal{D}_{target}$.

Our goal is to facilitate power modeling of these in-progress projects based on limited data of their target architectures $\mathcal{D}_{target}$. FirePower achieves this in two phases: knowledge extraction and application. In the knowledge extraction phase, the developer extracts generalizable knowledge based on known architecture $\mathcal{D}_{known}$. In the application phase, the knowledge is generalized to help develop power models for target architectures based on $\mathcal{D}_{target}$.

4 METHODOLOGY

4.1 Power Model Overview

The general FirePower solution is based on two basic insights.

Insight 1. Instead of directly modeling total power, architecture-level power should be modeled for each *power-friendly* component. Specifically, the whole design will be partitioned into multiple common components for power modeling purposes. Individual power models will be developed for each component, and total power is a summation of all component power. This brings multiple benefits: 1) compared with total power, individual components altogether provide more power labels; 2) designers have the flexibility to define components in a power-friendly manner, which is introduced in Section 4.2; and 3) smaller common components are affected by very few hardware parameters, thus component power model tends to be simple thus more general. It is further discussed in Insight 2.

Insight 2. When developing data-driven architecture-level power models, we observe that some knowledge tends to be more general, while others are more architecture-specific. Here more general knowledge refers to correlation patterns between basic component *hardware parameters* H and component *hardware scale*, which describes the overall amount of logic (e.g., number of logic gates) in the component. One important reason is, the number of hardware parameters of each component is very limited, ranging from 1 to 4 in our experiment. Patterns based on fewer hardware parameters tend to be simpler and thus less "overfit" to a specific architecture. In contrast, knowledge related to event statistics E involves not only complex event activities but also the interaction between events and hardware scale. This makes patterns related to event statistics relatively complex. We thus set this part architecture-specific.

Inspired by two aforementioned insights, FirePower chooses com-ponent-level power modeling, with each component's power decoupled into generalizable and architecture-specific parts. With hardware parameters of the i-th component as H_i and event statistics as E_i, the FirePower power model can be formulated below.

$$P^i = \boldsymbol{F_{hw}}^i(H_i) * \boldsymbol{F_{event}}^i(H_i, E_i)$$

The $\boldsymbol{F_{hw}}^i$ denotes the hardware model of component i, which learns the basic correlation between hardware scale and hardware parameters. The $\boldsymbol{F_{event}}^i$ denotes the event model, an ML model to capture more complex correlations related to event statistics. As for why multiplication is used to associate these two models, power consumption P^i is roughly proportional to hardware scale (e.g., number of logic gates) in $\boldsymbol{F_{hw}}^i$, assuming a constant average toggle rate. The event-related knowledge captured by $\boldsymbol{F_{event}}^i$ partially reflects the toggle rate of the real workload. The toggle rate is also proportional to power. Multiplication is the simplest operator to capture the linear relationship between these two models and power.

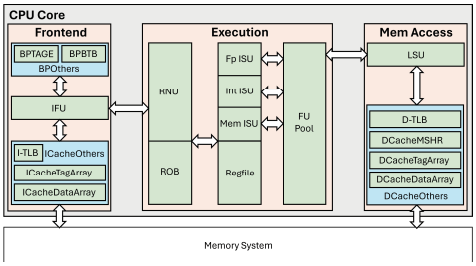


Figure 2: The illustration of our power-friendly component definition for the out-of-order CPU core.

Component i	Hardware Parameters of Each Component H_i	Important Parameter
BPTAGE	FetchWidth, BranchCount	FetchWidth
BPBTB	FetchWidth, BranchCount	FetchWidth
BPOthers	FetchWidth, BranchCount	FetchWidth
IFU	FetchWidth, DecodeWidth, FetchBufferEntry, ICacheFetchBytes	–
I-TLB	ICacheTLBEntry	–
ICacheTagArray	ICacheWay, ICacheFetchBytes	DCache/ICacheWay
ICacheDataArray	ICacheWay, ICacheFetchBytes	FetchWidth
ICacheOthers	ICacheWay, ICacheFetchBytes	–
RNU	DecodeWidth	DecodeWidth
ROB	DecodeWidth, RobEntry	–
FP ISU	DecodeWidth, FpIssueWidth	–
Int ISU	DecodeWidth, IntIssueWidth,	DecodeWidth
Mem ISU	DecodeWidth, MemIssueWidth	–
Regfile	DecodeWidth, IntPhyRegister, FpPhyRegister	–
FU Pool	Mem/FpIssueWidth, IntIssueWidth	Mem/FpIssueWidth
LSU	LDQEntry, STQEntry, MemIssueWidth	–
D-TLB	DCacheTLBEntry	DTLBEntry
DCacheTagArray	DCacheWay, DCacheTLBEntry, MemIssueWidth	–
DCacheDataArray	DCacheWay, DCacheTLBEntry, MemIssueWidth	–
DCacheMSHR	MSHREntry	MSHREntry
DCacheOthers	DCacheWay, DCacheTLBEntry, MSHREntry, MemIssueWidth	–
Other Logic	All	–

Table 1: Our identified architecture-level hardware parameters and the important parameter detected for Retraining.

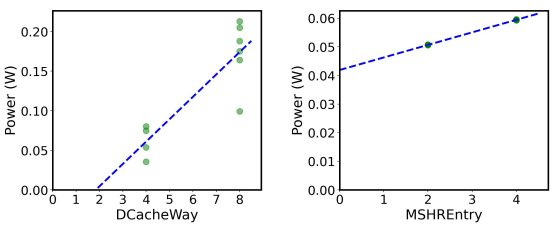
The remainder of this section will introduce the FirePower methodology in detail. Section 4.2 describes our proposed power-friendly component definition for power modeling at the component level. Section 4.3 and Section 4.4 will introduce the knowledge extraction (Phase 1) and application (Phase 2) of FirePower, respectively.

4.2 Power-Friendly Component Definition

To facilitate component-level data-driven power modeling, we propose a power-friendly component definition for out-of-order CPU. The component definition has two targets: being common and fine-grained. 1) To generalize power model to different microarchitecture designs, the component definition should be common, where each component can be found in different out-of-order CPUs. 2) To facilitate per-component power modeling, component definition should be fine-grained so that the circuit in the same component should correlate with similar hardware parameters and event statistics.

To meet the two targets above, we propose a power-friendly component definition. The components are in three main parts: Frontend, Execution, and Mem Access. Details are introduced below.

- The Frontend includes 8 components: TAGE in branch predictor (BPTAGE), BTB in branch predictor (BPBTB), others in branch predictor (BPOthers), instruction fetch unit (IFU), instruction translation lookup buffer (I-TLB), instruction cache tag array (ICacheTagArray), instruction cache data array (ICacheDataArray), and others in instruction cache (ICacheOthers).



(a) DCacheDataArray (b) DCacheMSHR

Figure 3: Correlation between power and the most related hardware parameter. The two components correlate with different hardware parameters. DCacheDataArray correlates with *DCacheWay*, DCacheMSHR correlates with *MSHREntry*.

- The Execution consists of 7 components: renaming unit (RNU), reorder buffer (ROB), issue unit of float point instruction (Fp ISU), issue unit of integer instruction (Int ISU), issue unit of memory access instruction (Mem ISU), register file (Regfile), and function unit pool (FU Pool).
- The Mem Access has 6 components: load-store unit (LSU), data translation lookup buffer (D-TLB), miss status handle register (DCacheMSHR), data cache tag array (DCacheTagArray), data cache data array (DCacheDataArray), and others in data cache (DCacheOthers).
- Other circuits not covered by the above components are referred to as a new component named Other Logic.

Table 1 shows the associated hardware parameters of each component, where event statistics are not listed because of page limitation.

No existing architecture-level ML power modeling works built component-level models except PANDA [32]. The component power model in PANDA [32] is based on default component partitioning without considering power. For example, in [32], the DCache is a whole component, but within this DCache, we can find that the DCacheDataArray highly correlates with a hardware parameter *DCacheWay*, while DCacheMSHR correlates with another hardware parameter *MSHREntry*. The correlation is illustrated in Fig. 3 using the XiangShan CPU [29], showing their correlations with different hardware parameters. They are thus separated and processed with different power models in FirePower to capture clearer patterns. It conforms to the aforementioned fine-granularity target.

4.3 Phase 1: Knowledge Extraction

In phase 1 of FirePower, developers perform knowledge extraction, extracting the generalizable knowledge based on sufficient data of an already known architecture. This process only needs to be performed once by solution developers, as shown in the left of Fig. 4. Based on the known architecture, generalizable knowledge is extracted for each component, including two types of information: (1) hardware model F^i_{hw} built on the known architecture, (2) the importance of the hardware parameters of this component.

4.3.1 Hardware Model. The hardware model $\boldsymbol{F_{hw}}^i$ learns the relationship between hardware scale and hardware parameters, as introduced in the overview. To represent the hardware scale as labels, we calculate the average power across all workloads. This average power label reflects the general power characteristics across workloads. The input features, as summarized in Table 1, are hardware parameters H_i of each component. The ML model we use is the XGBoost [10], which is one of the most widely adopted regressors.

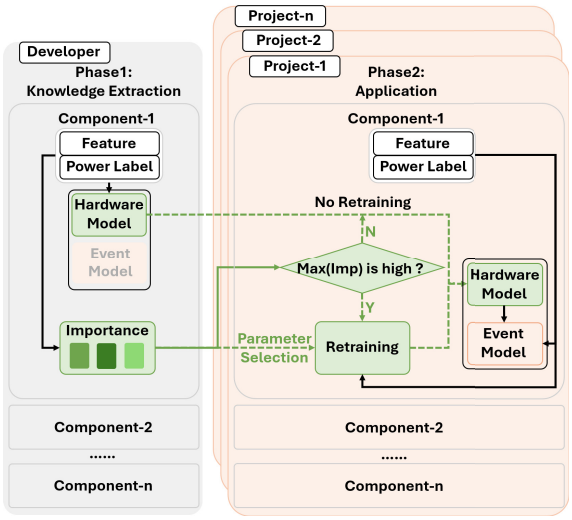


Figure 4: The FirePower framework with two phases. Knowledge extraction in phase 1 extracts hardware model and parameter importance from a known architecture as general knowledge. Application in phase 2 adopts two knowledge generalization strategies, Retraining and No Retraining, depending on the parameter importance distribution.

It is a decision-tree-based ensemble learning algorithm using an ensemble of weak prediction models for regression.

4.3.2 Parameter Importance. In addition to the hardware model, we will further evaluate the *importance* of hardware parameters H_i for each component. Such importance reflects the impact of each hardware parameter on the component power. The distribution of parameter importance will help us analyze the power correlation pattern for each component: The key idea is to evaluate whether there is a dominating hardware parameter for each component. It will affect how to generalize the knowledge in phase 2. More details about the usage of parameter importance are discussed in Sec. 4.4.1.

The hardware parameter importance is calculated based on the hardware model. Such a tree-model-based evaluation calculates feature importance based on impurity decreases contributed by each feature (e.g., parameter) [8]. Importance evaluation is not limited to tree models, there are also some methods, such as SHAP [21], to evaluate parameter importance for arbitrary ML models.

4.4 Phase 2: Application

The application as phase 2 of FirePower is shown in the right of Fig. 4. Compared to knowledge extraction, which is performed only once in total by the developer, the application phase can be applied many times to different projects and target architectures. In each project, phase 2 utilizes the extracted generalizable knowledge and the limited data of the target architecture to build the power model for the target architecture. Specifically, the application phase performs three steps: (1) apply the generalizable knowledge to build the hardware model $\boldsymbol{F_{hw}}^i$, (2) train the event model $\boldsymbol{F^i_{event}}$, and (3) evaluate the generalization to estimate the generalization's quality.

4.4.1 Hardware Model. For each component, we support two knowledge generalization strategies, named Retraining and No Retraining, as illustrated in Fig. 4.

Strategy 1: No Retraining is straightforward, it directly adopts hardware model ${F_{hw}}^i$ trained on known architecture dataset $\mathcal{D}_{known}$ for target one. Directly applying hardware model may not accurately estimate average power due to differences in hardware scale of different architectures. The rationale behind No Retraining is hardware model primarily captures the correlation rather than the absolute average power value. The ratio between known and target architecture will be captured by additional event model F^i_{event}.

Strategy 2: For Retraining, it trains a brand new hardware model using limited available configurations $\mathcal{D}_{target}$ of the target architecture. Such a retraining faces a trade-off. On the one hand, available configurations directly from target architecture naturally help transfer. On the other hand, since available configurations in $\mathcal{D}_{target}$ are limited, the model can easily overfit. To avoid overfitting, the key idea behind retraining is to maximally simplify new hardware model ${F_{hw}}^i$. This retrained hardware model ${F_{hw}}^i$ will be a linear model based on one most important parameter from H_i. Retraining policy is more suitable for components with relatively simple power characteristics with one dominating important parameter.

Strategy selection: For each component, we select the most appropriate strategy (Retraining vs. No Retraining) based on parameter importance distribution. Such parameter importance is also the knowledge extracted from known architecture in phase 1. When only one hardware parameter has a dominating importance, it indicates the overall power correlation is simple. On the contrary, if the distribution is more uniform, it means this component is relatively complex. Therefore, if maximum parameter importance exceeds a threshold[2], Retraining will be adopted. Otherwise, No Retraining is selected. The strategy selection result based on BOOM CPU architecture is listed in Table 1, where the important parameter used for Retraining is listed, and the "–" means No Retraining.

4.4.2 Event Model. The event model F^i_{event} will mainly capture the more complex correlation related to event statistics, which is highly architecture-specific. To train the event model for target architectures, for each component, we take both hardware parameters H_i and event statistics E_i of each component as features. The event model's training label is P_i/F^i_H, which is the ratio between the component power label in $\mathcal{D}_{target}$ and the hardware model F^i_H. The adopted ML model is also XGBoost [10].

4.5 Generalization Quality Evaluation

The effectiveness of knowledge generalization can be compromised when there is a significant difference between the target architecture and the known architecture. Hence, it is crucial to evaluate the generalization's quality to help determine whether to accept generalized power model or resort to time-consuming traditional power modeling paradigm. Such quality evaluation helps indicate *applicable scope* of FirePower based on the known architecture.

For such generalization evaluation on each component, we compare average power labels of each target configuration from $\mathcal{D}_{target}$ with predictions of hardware model F^i_{hw} from phase 1, without considering retraining. Such F^i_{hw} reflects known architecture $\mathcal{D}_{known}$, and thus the comparison indicates the difference between known and target architecture. When comparing, we adjust prediction of hardware model by multiplying it with an ideal scaling factor, since hardware model F^i_{hw} only captures the trend and detailed ratios are left for event models F^i_{event}, as discussed in Sec. 4.4.1.

[2]We set the threshold to 0.95 in the experiment, with the sum of all hardware parameters H_i in each component normalized to 1.

5 EXPERIMENT SETUP

5.1 RISC-V CPU Cores for Experiment

In our experiment, we adopt two different RISC-V CPU cores as our experimented architectures. RISC-V [5] is one of the most widely adopted open-source instruction set architecture. Nowadays, most open-source CPU design projects are based on RISC-V, the most representative two projects of which are BOOM [33] and XiangShan [29]. BOOM and XiangShan are both highly configurable, enabling us to generate different configurations for each architecture. They are both out-of-order CPU cores with similar major CPU components but there are also many differences. Although they both use RISC-V, the version is not the same, with RV64GC for BOOM and RV64GCBK for XiangShan. Besides, architectural designs for components are different. Taking the second-level branch predictor as an example, BOOM adopts traditional BTB for target prediction, while XiangShan adopts Fetch Target Buffer to replace BTB. Considering the reasonable similarities and differences between BOOM and XiangShan, we evaluate our paradigm on them.

When evaluating each method, we conduct multiple experiments with different known/target architecture settings and different numbers of configurations of target architecture. Because of limited accessible open-source RISC-V CPU architectures, there is only one target architecture. Known/target architecture settings include BOOM as known architecture and XiangShan as target architecture denoted as BOOM → XS and vice versa denoted as XS → BOOM. For different numbers of configurations of target architecture, in knowledge extraction (Phase 1), all known architecture configurations are used. For application (Phase 2), configurations of target architecture are used for knowledge generalization, and remaining ones are used for testing. We evaluate the accuracy with mean absolute percentage error (MAPE) and correlation coefficient R.

Configurations adopted for BOOM and XiangShan are listed in Table 2, covering different scales. There are 15 configurations for BOOM named B1 to B15 and 10 for XiangShan named X1 to X10. To reflect the scenario where only a few configurations of target architecture are accessible, the number of labeled configurations of target architecture is set as 4, 3, and 2 for three sets of experiments.

5.2 Design Implementation Flow

In our experiment, to collect the dataset, RTL code generation and RTL simulation of BOOM CPU [33] is performed with Chipyard [6] v1.8.1, and that of XiangShan CPU [29] is performed with OpenXiangShan [4]. For workload-driven power simulation to generate ground truth power, we used eight workloads in riscv-tests [3] including dhrystone, median, multiply, qsort, rsort, towers, spmv, and vvadd. Minor modifications are made for adaptation on XiangShan.

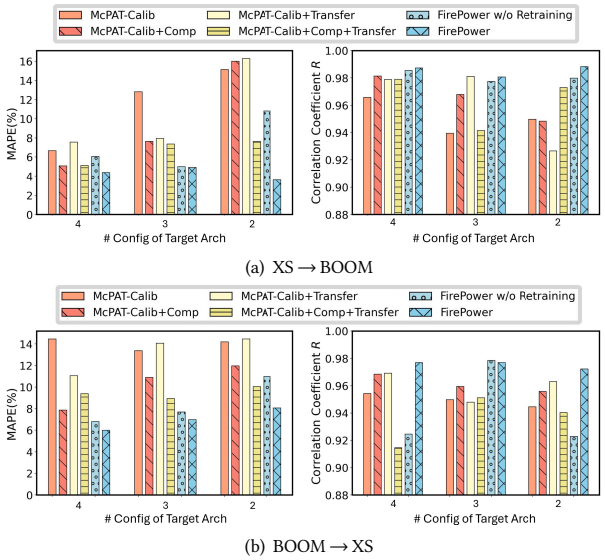
The RTL simulation is performed with Synopsys VCS® [2]. We performed logic synthesis and power simulation with Synopsis Design Compiler® [1] and PrimePower [24] respectively. Our VLSI flow is based on TSMC 40nm standard cell library and associated Memory Compiler for SRAM generation. For the microarchitecture simulation, we use gem5 [7] as performance simulator to generate event statistics. We also use McPAT [19] as power model to generate power estimation as some of features for McPAT-Calib.

Hardware Parameter	B1	B2	B3	B4	B5	B6	B7	B8	B9	B10	B11	B12	B13	B14	B15	X1	X2	X3	X4	X5	X6	X7	X8	X9	X10
FetchWidth	4	4	4	4	4	8	8	8	8	8	8	8	8	8	8	4	4	4	4	4	8	8	8	8	8
DecodeWidth	1	1	1	2	2	2	3	3	3	4	4	4	5	5	5	2	2	2	3	3	3	4	4	4	5
FetchBufferEntry	5	8	16	8	16	24	18	24	30	24	32	40	30	35	40	8	16	24	16	24	24	24	32	32	24
RobEntry	16	32	48	64	64	80	81	96	114	112	128	136	125	130	140	16	32	48	64	64	80	81	96	114	112
IntPhyRegister	36	53	68	64	80	88	88	110	112	108	128	136	108	128	140	36	53	68	64	80	88	88	110	112	108
FpPhyRegister	36	48	56	56	64	72	88	96	112	108	128	136	108	128	140	36	53	68	64	80	88	88	110	112	108
LDQ/STQEntry	4	8	16	12	16	20	16	24	32	24	32	36	24	32	36	16	20	24	20	24	28	24	32	40	32
BranchCount	6	8	10	10	12	14	14	16	16	18	20	20	18	20	20	7	7	7	7	7	7	7	7	7	7
Mem/FpIssueWidth	1	1	1	1	1	1	1	1	2	1	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2
IntIssueWidth	1	1	1	1	2	2	2	3	3	4	4	4	5	5	5	2	2	2	2	4	4	4	6	6	6
DCache/ICacheWay	2	4	8	4	4	8	8	8	8	8	8	8	8	8	8	4	4	8	4	4	8	8	8	8	8
DTLBEntry	8	8	16	8	8	16	16	16	32	32	32	32	32	32	32	8	8	16	8	8	16	16	16	32	32
MSHREntry	2	2	4	2	2	4	4	4	4	4	4	8	8	8	8	2	2	4	2	2	4	4	4	4	4
ICacheFetchBytes	2	2	2	2	2	4	4	4	4	4	4	4	4	4	4	2	2	2	2	2	2	2	2	2	2

Table 2: The CPU configurations used in our experiment. The B1-B15 denote the 15 configurations of BOOM, and the X1-X10 denote the 10 configurations of XiangShan.

5.3 Summary of Baseline Methods

We compare FirePower with the state-of-the-art architecture-level power model McPAT-Calib [30] as our baseline. The other work [32] is not included since it is not an automated method, requiring engineer-defined functions. Besides McPAT-Calib [30], we further include four more ablation studies based on part of FirePower's policies. (1) The McPAT-Calib + Component. It builds ML models for each component, using the associated hardware parameters and event statistics as features and per-component power as labels. (2) The McPAT-Calib + Transfer Learning. It adopts the McPAT-Calib as the power modeling method, builds a model on known architecture as the source model, and then adopts one of the most widely adopted transfer learning algorithms, pseudo label [17], for knowledge generalization. In detail, for each testing data of the target architecture, we search for the nearest labeled data of the target architecture using the distance in feature space, where the label is L. We use the source model to make predictions for the testing data and its nearest sample, denoted as p_t and p_l. The prediction on test data is $P_t = \frac{p_t}{p_l} L$. (3) The McPAT-Calib + Component + Transfer Learning. It combines (1) and (2), performing transfer learning for each component respectively. (4) FirePower without Retraining. It only adopts the No Retraining as the knowledge generalization strategy, without taking the parameter importance as the generalizable knowledge. For a fair comparison, for all baselines and our FirePower solution, we adopt the same XGBoost [10], which is the best ML model reported in McPAT-Calib [30], with default hyperparameters, i.e. n_estimator=100 and depth=3.



(a) XS → BOOM

(b) BOOM → XS

Figure 5: Summary of the comparison between FirePower and other methods under different numbers of configurations of target architecture. "Comp" stands for Component and "Transfer" stands for Transfer Learning.

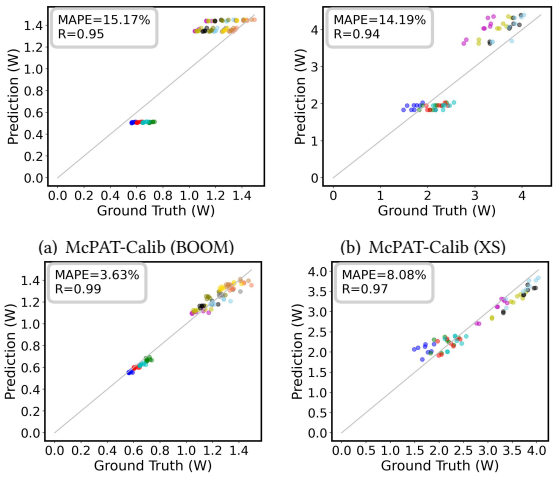
6 EXPERIMENTAL RESULTS

6.1 Power Modeling Accuracy

Fig. 5 summarizes comparisons between FirePower with our baseline, McPAT-Calib, and four ablation studies, under different numbers of available configurations of target architecture, i.e. 4, 3, and 2 configurations. Fig. 6 further visualize detailed results for FirePower and McPAT-Calib with only 2 available configurations, where samples of the same configuration are in the same color. The comparison with McPAT-Calib shows that FirePower can consistently achieve superior accuracy over McPAT-Calib regardless of scenarios. FirePower achieves the lowest MAPE and the highest correlation coefficient R, with at most (on average) 11.5% (7%) lower MAPE and 0.04 (0.03) higher correlation R compared with McPAT-Calib. With only two configurations of target architecture, FirePower can still achieve a low MAPE of 5.8% and a high correlation R of 0.98 on average, which is 8.8% lower in error percentage and 0.03 higher in R compared with McPAT-Calib. The superiority of FirePower over McPAT-Calib is contributed by its ability to generalize knowledge acquired from known architecture. In contrast, architecture-specific McPAT-Calib trains model from scratch.

Fig. 5 also shows FirePower can constantly achieve the best accuracy compared with four ablation studies for both MAPE and correlation R. McPAT-Calib + Component is an enhanced version of McPAT-Calib by building models for each component. It has an advantage over McPAT-Calib, validating the effect of power-friendly component definition. However, it can still not achieve a high accuracy compared with FirePower. It verifies knowledge generalization is critical to enable few-shot power modeling.

McPAT-Calib + Transfer Learning and McPAT-Calib + Component + Transfer Learning are two knowledge generalization methods based on transfer learning. They directly transfer the power model as a whole, regardless of generality. Results in Fig. 5 show that they outperform McPAT-Calib and McPAT-Calib + Component in many scenarios. It demonstrates that information from other known architecture can improve accuracy of few-shot modeling. But FirePower still has an obvious advantage over them,



(a) McPAT-Calib (BOOM) (b) McPAT-Calib (XS)

(c) FirePower (XS → BOOM) (d) FirePower (BOOM → XS)

Figure 6: Accuracy comparison between FirePower and McPAT-Calib (Available Config of Target Arch = 2).

which is because these two methods do not decouple general and architecture-specific knowledge. It verifies the necessity of decoupling generalizable and architecture-specific knowledge, where generalizing architecture-specific knowledge has negative impact.

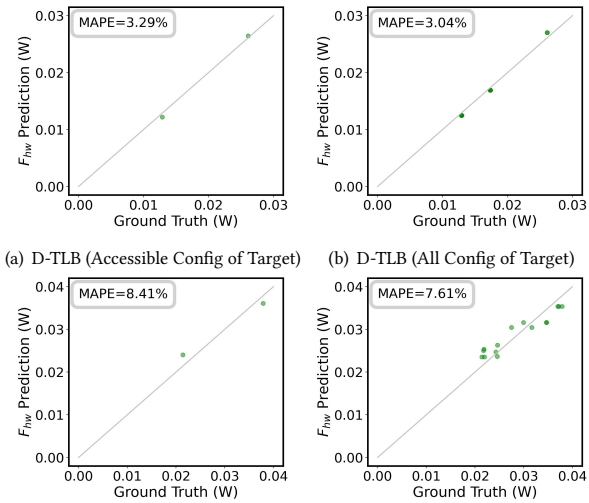
Fig. 5 also shows that "FirePower without Retraining" is more accurate than McPAT-Calib + Component, also outperforming two transfer-learning-based methods in some scenarios. This validates that even generalization with No Retraining can also work. Comparison between FirePower and FirePower without Retraining verifies that the Retraining and strategy selection are crucial. The parameter importance is essential generalizable knowledge for FirePower.

6.2 Generalization Evaluation

Fig. 7 illustrates the generalization evaluation for components with high similarity between known and target architecture. Fig. 8 shows the lower-similarity one. Higher similarity is supposed to result in a high generalization quality, and vice versa.

In Fig. 7 and 8, each point represents a configuration of the target architecture. The x-axis is the golden average power across workloads. The y-axis is the adjusted prediction of the hardware model trained on known architecture, which is discussed in Sec. 4.5. The similarity between these two values indicates generalization quality, which can be measured with MAPE. Fig. 7(b)(d) and Fig. 8(b) show actual generalization quality with all configurations of the target architecture, which is not accessible when building the model. Fig. 7(a)(c) and Fig. 8(a) show generalization quality that we can observe with limited accessible configurations of target architecture.

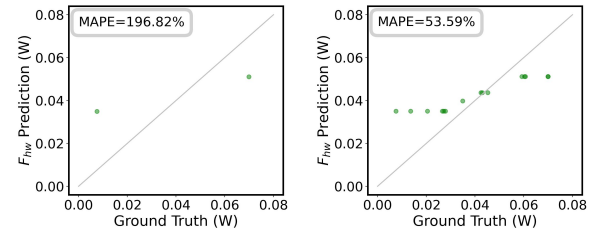
We can find the actual generalization quality evaluated with all configurations in Fig. 7(b)(d) and Fig. 8(b) correlates with generalization quality measured with the accessible ones in Fig. 7(a)(c) and Fig. 8(a). This means architects can roughly estimate the quality with the generalization quality observed with the accessible configurations. Generally, as illustrated in these figures, if the generalization quality observed with the accessible configurations has a MAPE lower than 10%, it suggests a relatively high actual generalization quality indicating that it can result in a high-quality generalized model. Conversely, if MAPE exceeds this threshold, the



(a) D-TLB (Accessible Config of Target) (b) D-TLB (All Config of Target)

(c) OtherLogic (Accessible Config of Target) (d) OtherLogic (All Config of Target)

Figure 7: Generalization quality evaluation for components with high similarity across architectures. (a)(c) The generalization qualities observed with the accessible configuration of the target architecture. (b)(d) The generalization qualities evaluated with all configurations of the target architecture.



(a) LSU (Accessible Config of Target) (b) LSU (All Config of Target)

Figure 8: Generalization quality evaluation for components with low similarity across architectures. (a) The generalization quality observed with the accessible configuration of the target architecture. (b) The generalization quality evaluated with all configurations of the target architecture.

generalization may result in a low-quality model. In experiments, we always adopt generalized hardware model, because power percentage of evaluated low-similarity components is relatively small.

7 CONCLUSION

We propose FirePower which targets few-shot learning scenario for new target architectures by different users. Developer extracts the generalizable knowledge from a well-developed architecture, and then multiple projects can use this knowledge for few-shot power modeling, with limited available configurations of target architectures. The foundation-based paradigm reduces data requirement significantly, which is a compelling addition to architects' toolbox.

ACKNOWLEDGEMENT

This work is partially supported by National Natural Science Foundation of China 62304192, and ACCESS – AI Chip Center for Emerging Smart Systems, sponsored by InnoHK funding, Hong Kong SAR. We acknowledge the suggestions from Dr. Andrea Mondelli.

REFERENCES

[1] 2021. Design Compiler® RTL Synthesis. https://www.synopsys.com/implementation-and-signoff/rtl-synthesis-test/design-compiler-nxt.html.

[2] 2021. VCS® functional verification solution. https://www.synopsys.com/verification/simulation/vcs.html.

[3] 2022. *RISC-V Tests*. https://github.com/riscv-software-src/riscv-tests.

[4] 2023. *OpenXiangShan*. https://github.com/OpenXiangShan.

[5] 2023. *RISC-V*. https://riscv.org.

[6] Alon Amid, David Biancolin, Abraham Gonzalez, Daniel Grubb, Sagar Karandikar, Harrison Liew, Albert Magyar, Howard Mao, Albert Ou, Nathan Pemberton, et al. 2020. Chipyard: Integrated design, simulation, and implementation framework for custom socs. *IEEE Micro* 40, 4 (2020), 10–21.

[7] Nathan Binkert, Bradford Beckmann, Gabriel Black, Steven K Reinhardt, Ali Saidi, Arkaprava Basu, Joel Hestness, Derek R Hower, Tushar Krishna, Somayeh Sardashti, et al. 2011. The gem5 simulator. *ACM SIGARCH computer architecture news* 39, 2 (2011), 1–7.

[8] Leo Breiman. 2001. Random forests. *Machine learning* (2001).

[9] David Brooks, Vivek Tiwari, and Margaret Martonosi. 2000. Wattch: A framework for architectural-level power analysis and optimizations. *ACM SIGARCH Computer Architecture News* 28, 2 (2000), 83–94.

[10] Tianqi Chen and Carlos Guestrin. 2016. Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*. 785–794.

[11] Wenji Fang, Yao Lu, Shang Liu, Qijun Zhang, Ceyu Xu, Lisa Wu Wills, Hongce Zhang, and Zhiyao Xie. 2023. MasterRTL: A Pre-Synthesis PPA Estimation Framework for Any RTL Design. In *2023 IEEE/ACM International Conference on Computer Aided Design (ICCAD)*. IEEE, 1–9.

[12] Wenji Fang, Yao Lu, Shang Liu, Qijun Zhang, Ceyu Xu, Lisa Wu Wills, Hongce Zhang, and Zhiyao Xie. 2024. Transferable Pre-Synthesis PPA Estimation for RTL Designs With Data Augmentation Techniques. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* (2024).

[13] Abdullah Guler and Niraj K Jha. 2020. 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* 28, 10 (2020), 2146–2156.

[14] Donggyu Kim, Jerry Zhao, Jonathan Bachrach, and Krste Asanović. 2019. Simmani: Runtime power modeling for arbitrary RTL with automatic signal selection. In *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture*. 1050–1062.

[15] Ajay Krishna Ananda Kumar, Sami Al-Salamin, Hussam Amrouch, and Andreas Gerstlauer. 2022. Machine learning-based microarchitecture-level power modeling of CPUs. *IEEE Trans. Comput.* 72, 4 (2022), 941–956.

[16] Ajay Krishna Ananda Kumar and Andreas Gerstlauer. 2019. Learning-based CPU power modeling. In *2019 ACM/IEEE 1st Workshop on Machine Learning for CAD (MLCAD)*. IEEE, 1–6.

[17] Dong-Hyun Lee et al. 2013. Pseudo-label: The simple and efficient semi-supervised learning method for deep neural networks. In *Workshop on challenges in representation learning, ICML*, Vol. 3. Atlanta, 896.

[18] Wooseok Lee, Youngchun Kim, Jee Ho Ryoo, Dam Sunwoo, Andreas Gerstlauer, and Lizy K John. 2015. PowerTrain: A learning-based calibration of McPAT power models. In *2015 IEEE/ACM International Symposium on Low Power Electronics and Design (ISLPED)*. IEEE, 189–194.

[19] Sheng Li, Jung Ho Ahn, Richard D Strong, Jay B Brockman, Dean M Tullsen, and Norman P Jouppi. 2009. McPAT: An integrated power, area, and timing modeling framework for multicore and manycore architectures. In *Proceedings of the 42nd annual ieee/acm international symposium on microarchitecture (MICRO)*. 469–480.

[20] Yao Lu, Qijun Zhang, and Zhiyao Xie. 2024. Unleashing Flexibility of ML-based Power Estimators Through Efficient Development Strategies. In *Proceedings of the 29th ACM/IEEE International Symposium on Low Power Electronics and Design*. 1–6.

[21] Scott M Lundberg and Su-In Lee. 2017. A unified approach to interpreting model predictions. *Advances in neural information processing systems* 30 (2017).

[22] Jian Peng, Tingyuan Liang, Zhiyao Xie, and Wei Zhang. 2023. PROPHET: Predictive On-Chip Power Meter in Hardware Accelerator for DNN. In *2023 60th ACM/IEEE Design Automation Conference (DAC)*. IEEE, 1–6.

[23] Siemens. 2023. PowerPro RTL Low-Power. https://www.mentor.com/hls-lp/powerpro-rtl-low-power/

[24] Synopsys. 2023. PrimePower: RTL to Signoff Power Analysis. https://www.synopsys.com/implementation-and-signoff/signoff/primepower.html

[25] Aoxiang Tang, Yang Yang, Chun-Yi Lee, and Niraj K Jha. 2014. McPAT-PVT: Delay and power modeling framework for FinFET processor architectures under PVT variations. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 23, 9 (2014), 1616–1627.

[26] Sam Likun Xi, Hans Jacobson, Pradip Bose, Gu-Yeon Wei, and David Brooks. 2015. Quantifying sources of error in McPAT and potential impacts on architectural studies. In *2015 IEEE 21st International symposium on high performance computer architecture (HPCA)*. IEEE, 577–589.

[27] Zhiyao Xie, Shiyu Li, Mingyuan Ma, Chen-Chia Chang, Jingyu Pan, Yiran Chen, and Jiang Hu. 2022. DEEP: Developing extremely efficient runtime on-chip power meters. In *Proceedings of the 41st IEEE/ACM International Conference on Computer-Aided Design*. 1–9.

[28] Zhiyao Xie, Xiaoqing Xu, Matt Walker, Joshua Knebel, Kumaraguru Palaniswamy, Nicolas Hebert, Jiang Hu, Huanrui Yang, Yiran Chen, and Shidhartha Das. 2021. APOLLO: An automated power modeling framework for runtime power introspection in high-volume commercial microprocessors. In *MICRO-54: 54th Annual IEEE/ACM International Symposium on Microarchitecture*. 1–14.

[29] Yinan Xu, Zihao Yu, Dan Tang, Guokai Chen, Lu Chen, Lingrui Gou, Yue Jin, Qianruo Li, Xin Li, Zuojun Li, et al. 2022. Towards developing high performance RISC-V processors using agile methodology. In *2022 55th IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 1178–1199.

[30] Jianwang Zhai, Chen Bai, Binwu Zhu, Yici Cai, Qiang Zhou, and Bei Yu. 2022. McPAT-Calib: A RISC-V BOOM microarchitecture power modeling framework. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)* 42, 1 (2022), 243–256.

[31] Jianwang Zhai, Yici Cai, and Bei Yu. 2023. Microarchitecture Power Modeling via Artificial Neural Network and Transfer Learning. In *2023 28th Asia and South Pacific Design Automation Conference (ASPDAC)*.

[32] Qijun Zhang, Shiyu Li, Guanglei Zhou, Jingyu Pan, Chen-Chia Chang, Yiran Chen, and Zhiyao Xie. 2023. PANDA: Architecture-level power evaluation by unifying analytical and machine learning solutions. In *2023 IEEE/ACM International Conference on Computer Aided Design (ICCAD)*. IEEE, 01–09.

[33] Jerry Zhao, Ben Korpan, Abraham Gonzalez, and Krste Asanovic. 2020. Sonicboom: The 3rd generation berkeley out-of-order machine. In *Fourth Workshop on Computer Architecture Research with RISC-V*, Vol. 5.

[34] Yuan Zhou, Haoxing Ren, Yanqing Zhang, Ben Keller, Brucek Khailany, and Zhiru Zhang. 2019. PRIMAL: Power inference using machine learning. In *Proceedings of the 56th Annual Design Automation Conference 2019*. 1–6.