Compilation Tells Energy: Rethinking Power Modeling for DNN Accelerator Agile Design

Donger Luo
ShanghaiTech University

Yihong Yi
ShanghaiTech University

Xinheng Li
ShanghaiTech University

Tianyi Li
ShanghaiTech University

Jianwang Zhai
Beijing University of Posts and Telecommunications

Qi Sun
Zhejiang University

Cheng Zhuo
Zhejiang University

Hao Geng†
ShanghaiTech University

Abstract

RTL generators enable agile design of DNN accelerators, but the lack of early-stage power feedback forces designers to discover energy inefficiencies only after costly synthesis. Existing arch-level simulator-based approaches fall short for agile workflows: they require expertise and effort incompatible with rapid iteration. While machine learning struggles to capture software-hardware coupling, we reveal a key insight: compilation tells energy. Compiler toolchains in RTL generators already fuse workload and hardware characteristics—the coupling determining power. By extracting features from compiler IRs and multi-task learning, our methodology achieves practical accuracy through push-button workflows requiring no expertise. Validation on Gemmini and hls4ml demonstrates broad applicability, enabling true power-aware agile design.

1 Introduction

The explosion of artificial intelligence across diverse scenarios, from various edge devices to data centers, demands scenario-specific DNN accelerators with vastly different power, performance, and area constraints. This drives agile design methodologies centered on RTL generators: Chisel-based frameworks like Gemmini [1] parameterize hardware templates for accelerators' configurable generation, while HLS-based tools like hls4ml [2] convert neural networks into C/C++ firmware, which is then synthesized into Verilog RTL via high-level synthesis for FPGA implementation. However, realizing their potential critically depends on early-stage power modeling. Power emerges from complex interactions between network characteristics (layer composition, operators, patterns) and hardware microarchitecture. Without accessible power feedback throughout iterative exploration, designers discover energy inefficiencies only after costly RTL synthesis and simulation, undermining the agility these methodologies promise.

Current power estimation approaches fall short for agile RTL generator workflows. As illustrated in Figure 1, architecture-level simulator-based methods (circled 1), like gem5 [3] or Timeloop /Accelergy [4, 5] and their ML-enhanced variants (circled 2) [6–10] require extensive hardware expertise and manual configuration, while suffering from prolonged runtimes. Simulation-free predictors offer

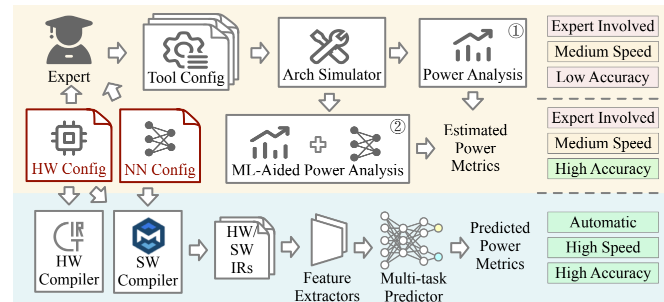

Figure 1: Architecture simulators and their ML-enhanced variants require expert-configured simulation, taking dozens of minutes, which is incompatible with agile iteration cycles. Our compilation-driven approach achieves high accuracy in seconds by extracting features from compiler IRs generated through push-button automated workflows, enabling rapid power feedback.

an automatic and fast alternative but face fundamental challenges: (1) diverse neural networks resist unified vectorization into common feature spaces; (2) parameterized hardware descriptions lose microarchitectural semantics when flattened into configuration vectors; (3) most critically, power emerges from intricate software-hardware coupling—which operations execute, how data flows through memory, what units activate.

Our key insight: compilers already do the hard work. RTL generators provide compilation toolchains that naturally fuse neural network workloads with hardware constraints through operator scheduling, memory allocation, and dataflow mapping. Unlike general-purpose CPUs executing arbitrary programs with unpredictable control flow, DNN accelerators run inference workloads with computational regularity—nested loops executing repeated kernels with predictable patterns. This regularity means compilation artifacts directly correlate with power consumption: compiled code structure serves as a reliable proxy for runtime power behavior.

As Figure 1 shows, we leverage this insight by extracting features from compiler intermediate representations and artifacts, where software-hardware interactions have already been systematically resolved. This transforms power modeling from learning arbitrary correlations between disparate specifications to learning the power implications of compiler-optimized execution patterns.

We instantiate this approach through a two-stage learning framework that strategically decouples training data requirements. For each compiler in the RTL generator toolchain, we encode verbose compilation outputs—potentially thousands of IR operations and

†Corresponding author, also with Shanghai Engineering Research Center of Energy Efficient and Custom AI IC.

This work is licensed under a Creative Commons Attribution 4.0 International License.
DAC '26, July 26–29, 2026, Long Beach, CA, USA
© 2026 Copyright held by the owner/author(s).
ACM ISBN 979-8-4007-2254-7/2026/07...$15.00
https://doi.org/10.1145/3770743.3804177

hardware elaboration details—into compact embeddings. This encoding is guided by pre-compilation features (network topology, hardware parameters) that enable effective compression while preserving power-relevant information. Since compilation artifacts are inexpensive to generate, we learn these encoding schemes from abundant pre/post-compilation pairs. The second stage then leverages these learned embeddings with limited but high-fidelity post-synthesis power data. We employ multi-task learning that jointly predicts average power and total energy consumption, allowing the model to exploit complementary signals where energy totals constrain time-integrated behavior while average power captures instantaneous resource utilization. This enables accurate power modeling with orders of magnitude less synthesis and simulation than end-to-end training approaches.

The main contributions of this work are:

- The first power prediction methodology for DNN accelerator RTL generators, leveraging compilation for software-hardware information fusion and feature extraction.
- A compilation-driven feature extraction approach that derives unified representations from compiler IRs and artifacts for diverse neural networks and hardware configurations.
- A two-stage learning framework decoupling abundant compilation data from limited RTL power data, with multi-task learning to maximize sample efficiency.
- Validation on Gemmini and hls4ml demonstrating that our approach achieves high prediction accuracy in agile design scenarios.

2 Preliminaries

2.1 Current Power Modeling Approaches

Architecture-level power modeling tools such as McPAT [11] and Accelergy [5] represent the mainstream methodology for estimating hardware power consumption. These tools follow a common workflow: they require detailed hardware configuration parameters (e.g., core specifications, memory hierarchy, NoC topology) combined with runtime statistics (e.g., access frequencies, resource utilization rates) typically extracted from architecture simulators like gem5 [3] or Timeloop [4]. Power estimates are then computed through analytical models or empirical formulas based on physical power equations. While widely adopted, these approaches suffer from accuracy limitations due to high-level abstractions that omit gate-level implementation details, and prolonged execution times dominated by prerequisite architecture simulation.

To address accuracy limitations, machine learning-enhanced power modeling methods have emerged [6, 7, 9, 10], including calibration of McPAT, optimization of analytical models, etc. However, these approaches remain fundamentally constrained by their dependence on architecture simulators: at inference time, they still require running tools like McPAT to extract input features. This dependency inherits the same usability challenges and runtime overhead.

Critically, these methodologies presume expert-driven workflows. Correctly configuring tools like Timeloop or gem5 demands deep architectural expertise—precisely specifying memory hierarchies, interconnect characteristics, and dataflow strategies for diverse accelerator designs. In traditional hardware design flows—where expert teams operate with month-long iteration cycles—these requirements are manageable. However, they fundamentally conflict with the agile development philosophy embodied by RTL generators, which prioritize rapid iteration (enabling efficient design space exploration) and democratized access (enabling algorithm engineers without hardware expertise to participate in co-design). This mismatch necessitates rethinking power modeling's information sources and computational pathways in an agile design flow.

2.2 Compilation in Agile Design Flow

RTL generators rely on compilation processes to bridge algorithmic intent and hardware realization. Compilation plays dual roles in these generators, operating on both software and hardware sides. Chisel-based generators like Gemmini employ two distinct compilation chains: a software compiler translating neural networks into accelerator-specific RISC-V binaries, and a hardware compiler elaborating parameterized Chisel descriptions through FIRRTL [12] intermediate representations (IRs) into synthesizable RTL. HLS-based tools like hls4ml leverage modern compilation infrastructures such as MLIR [13], which provide extensible multi-level IRs through domain-specific languages. These compilation processes perform critical transformations—operator scheduling, resource binding, memory allocation, and dataflow mapping—that inherently couple workload characteristics with hardware constraints.

Crucially, the IRs and artifacts generated during these compilation stages implicitly encode the coupled software-hardware execution semantics that determine power consumption. MLIR's progressive lowering—from high-level tensor operations through loop transformations and memory allocation to hardware-specific representations—exemplifies how compilation preserves and refines semantic information across abstraction levels. This observation motivates **our approach: rather than treating neural networks and hardware configurations as independent feature sources requiring models to rediscover their interactions, we extract features directly from compilation artifacts where this coupling has already been systematically performed and embedded into structured representations.**

2.3 Problem Formulation

We can formulate the problem of power modeling of DNN accelerator agile design as follows: Given a neural network $\mathcal{N}$, hardware configuration $\mathcal{H}$ (for parameterized accelerators). The objective is to predict power related metrics like average power and total energy with given $\mathcal{N}$ and $\mathcal{H}$, subject to the following constraints: (1) rapid inference, (2) push-button automation without expert configuration.

3 Compilation-based Power Prediction Flow

3.1 Why Compilation?

Power consumption fundamentally reflects dynamic runtime behavior rather than static specifications. It depends on which hardware units activate during execution, how data traverses memory hierarchies, and the order in which operations execute. In the scenario of RTL generators, while this information is implicitly determined by network descriptions and hardware parameters, deriving it requires analyzing their complex interactions—precisely what compilation already performs.

Compilation processes make these implicit behaviors explicit through concrete transformations. Software compilation maps abstract operators to instruction sequences, determining memory access patterns and data reuse strategies: for instance, identical convolutional layers can exhibit order-of-magnitude differences in SRAM accesses depending on loop tiling decisions made during compilation. Hardware compilation instantiates parameterized templates into actual datapaths, control logic, and buffer configurations: identical parameter settings may activate different hardware module combinations depending on the compiled workload. Critically, compiler

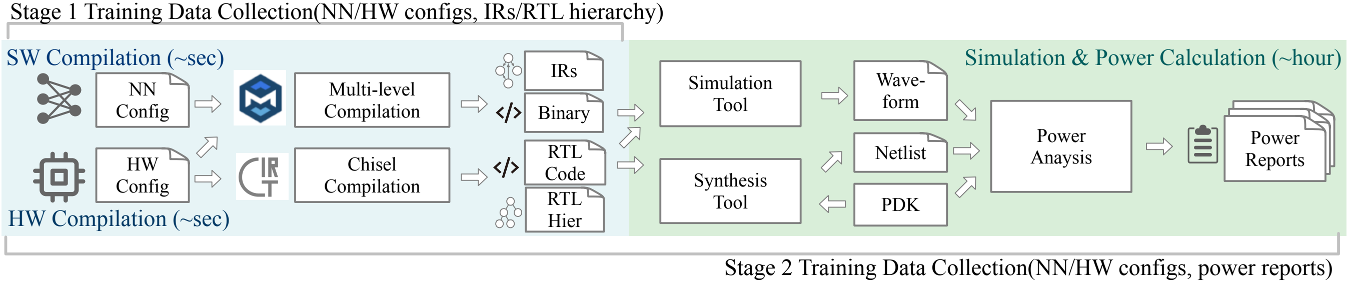

Figure 2: Overview of the traditional Chisel-based RTL generator's power calculation process: workload and hardware input is processed through the Chisel and multi-level compilation processes to train the feature extraction model, while simulation and power calculation results are used to train the power predictor.

intermediate representations capture the outcomes of these mapping decisions—the actual execution semantics—rather than merely encoding input specifications.

Beyond making power-relevant information explicit, compilation artifacts offer unique advantages for ML-based modeling.

- First, compilation provides **a unified representation space**: diverse neural network architectures—CNNs, Transformers, RNNs—are mapped through compilation into common IR operation sets and hardware instruction spaces, naturally resolving the heterogeneous network feature extraction challenge that plagues specification-based approaches.
- Second, IR graph and sequence structures preserve operation dependencies and dataflows in formats **amenable to modern feature extraction techniques**.
- Third, compilation workflows are **highly automated and require zero learning cost**—RTL generator vendors provide push-button compilation toolchains that execute without hardware expertise, contrasting sharply with architecture simulators that demand manual configuration like memory hierarchies and dataflow mappings.
- Finally, compilation-based features exhibit **compiler transparency**: when RTL generators upgrade their compiler optimizations, power models need not be retrained, as improved execution patterns are automatically reflected in the new compilation artifacts.

The challenge lies in efficiently extracting compact, power-relevant features from verbose compilation outputs—a problem we address through the two-stage learning framework described next.

3.2 Overall Flow

Power Model User Perspective (inference). As the lower part of Figure 1 shows, users provide neural network descriptions and hardware configuration parameters (if exist) in the generator's native format. These inputs pass through the generator's existing compilation processes, which may be single-stage or multi-stage depending on the specific platform. Pre-trained encoders extract features from the resulting compilation artifacts and produce embeddings that feed into a power predictor to output power metrics. This workflow operates transparently across different generator architectures, requiring neither simulation nor configuration expertise.

Power Model Provider Perspective (training). Our methodology adopts a two-stage training strategy that generalizes across diverse compilation architectures. Figure 2 shows the traditional power calculation flow, which is used as training data collection in our training strategy. **Stage 1 (Compilation Feature Extraction)** trains dedicated encoders for each compilation chain present in the target generator, compressing verbose compilation artifacts into compact embeddings using low-cost compilation data. **Stage 2 (Power Prediction)** trains a predictor that operates on concatenated encoder embeddings, employing multi-task learning to jointly predict power metrics using limited post-synthesis power data. This decoupled design accommodates different generator architectures—whether they employ separate software/hardware compilation chains (Chisel-based) or unified progressive lowering (HLS-based)—while maximizing data efficiency.

4 Methods

4.1 Software Compiler Feature Extraction

Compilation Artifacts and Information. Software compilation stages in RTL generators—whether targeting accelerator-specific instruction sets or progressive IR lowering in HLS frameworks—produce structured outputs encoding execution semantics. Regardless of specific instantiation, we focus on late-stage compilation outputs which offer critical advantages: (1) they expose accelerator-specific operations dominating power consumption, (2) provide stable representations across compiler front-end updates without requiring model retraining, and (3) create uniform feature spaces across diverse network architectures.

Graph Construction. We construct a computation graph $G_{\text{sw}} = (V, E)$ from compilation artifacts. Nodes correspond to functions or operator blocks partitioned at natural boundaries, while edges $e \in E$ capture control flow or data dependencies. This graph structure segments verbose compilation outputs into manageable subgraphs suitable for graph neural network processing while preserving execution dependencies.

Graph nodes are represented as compact statistical summaries. To manage vocabulary explosion from diverse instruction sets or operator types, we categorize atomic operations into $K \approx$ 10-20 functional classes based on likely hardware execution units, so operations within each category execute on shared resources and exhibit comparable energy characteristics. Each node v is encoded as a histogram vector $\mathbf{x}_v \in \mathbb{R}^K$ counting functional category occurrences, capturing the computational characteristics of each block.

Self-supervised Encoder Training. The upper-left part of Figure 3 shows the training strategy. To display the effectiveness of the proposed methodology, we employ a classic graph convolutional network (GCN) [14] encoder GCN_{sw}. Since compilation artifacts lack explicit supervision for power-relevant features, we use self-supervised pre-training through reconstruction tasks. The GCN encoder produces embeddings: $\mathbf{h}_{\text{sw}} = \text{GCN}_{\text{sw}}(G_{\text{sw}})$.

For Chisel-based generators with separate software and hardware compilation chains, software compilation is influenced by

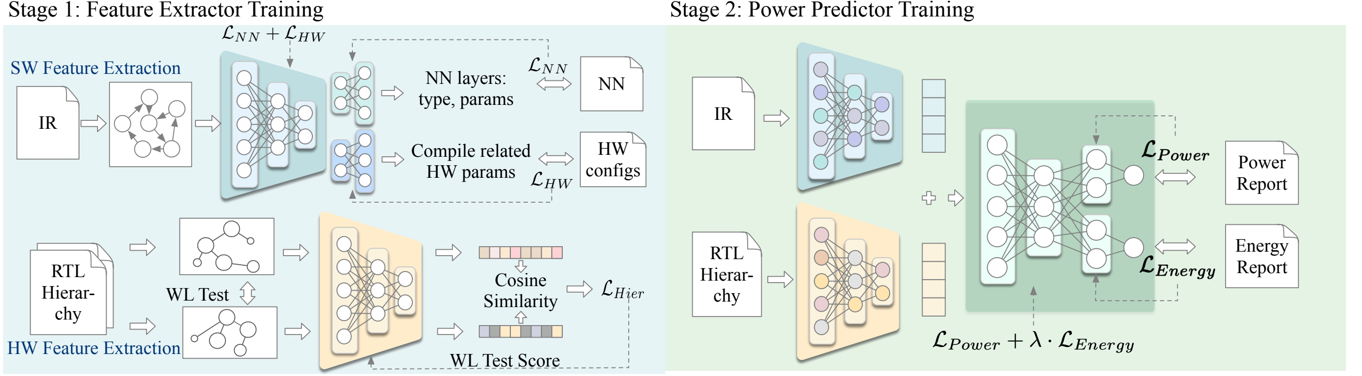

Figure 3: Two-Stage training flow of the Power Prediction model in the example of Chisel-based RTL generator: In stage 1, two GNN encoders for Hardware and Software Compiler feature extraction are pretrained using the feature information. Stage 2 uses the simulation and power calculation results, along with the pretrained model, to train the Power Predictor.

computation-relevant hardware parameters. We employ dual Transformer [15] decoders:

$$\hat{A} = \text{Decoder}_{\text{Net}}(\mathbf{h}_{\text{sw}}), \quad \hat{H} = \text{Decoder}_{\text{hw}}(\mathbf{h}_{\text{sw}}), \tag{1}$$

where $\hat{A}$ represents predicted network architecture as a variable-length sequence. Each element within the sequence encodes layer type and parameter count. Since network depth varies across workloads, we use Transformer decoders for autoregressive layer information generation. $\hat{H}$ represents predicted hardware parameters that participate in compilation. The network decoder forces the encoder to learn workload characteristics, while the hardware parameter decoder ensures embeddings capture how hardware configuration shapes compiled code. The training objective is:

$$\mathcal{L}_{\text{sw}} = \mathcal{L}_{\text{NN}} + \mathcal{L}_{\text{HW}} = -\sum_{t=1}^{T_A} \log p(a_t | a_{<t}, \mathbf{h}_{\text{sw}}) - \sum_{t=1}^{T_H} \log p(h_t | h_{<t}, \mathbf{h}_{\text{sw}}), \tag{2}$$

where $\{a_1, \ldots, a_{T_A}\}$ and $\{h_1, \ldots, h_{T_H}\}$ are network and hardware parameter token sequences.

For HLS-based generators with unified compilation flows, software compilation already encodes hardware mapping decisions through progressive lowering. We employ a single decoder:

$$\hat{A} = \text{Decoder}_{\text{Net}}(\mathbf{h}_{\text{sw}}), \tag{3}$$

with training objective:

$$\mathcal{L}_{\text{sw}} = \mathcal{L}_{\text{NN}} = -\sum_{t=1}^{T_A} \log p(a_t | a_{<t}, \mathbf{h}_{\text{sw}}). \tag{4}$$

Critically, both approaches, as illustrated in Figure 2, require only abundant compilation data without expensive RTL simulation.

4.2 Hardware Compiler Feature Extraction

RTL Hierarchy Trees and Structural Information. This subsection focuses on Chisel-based RTL generators where parameterized hardware is elaborated independently from the workload. For HLS-based flows that compile networks end-to-end to workload-specific RTL, software compilation artifacts already capture coupled software-hardware semantics, making separate hardware feature extraction redundant. Chisel compilation generates RTL hierarchy trees $G_{\text{hw}} = (V_{\text{hw}}, E_{\text{hw}})$, where nodes represent hardware modules and edges denote instantiation relationships. Unlike flattened parameter vectors that discard compositional information, hierarchy trees preserve RTL implementation semantics [16].

Contrastive Learning for Structural Encoding. The bottom-left part of Figure 3 shows the training strategy. Unlike computation graphs with natural reconstruction targets (network architecture and hardware parameters that shaped compilation), hierarchy trees lack obvious supervision signals since hardware configurations are already specified as input parameters—there is nothing to "reconstruct". We instead employ contrastive learning based on structural similarity, leveraging the insight that RTL trees with higher graph similarity—indicating analogous module composition and connectivity patterns—should exhibit similar power characteristics as they activate comparable hardware resources. This structural prior provides supervision without requiring expensive power measurements.

Specifically, we use Weisfeiler-Lehman (WL) graph kernel similarity $\text{sim}_{\text{WL}}(G_i, G_j)$ to identify structure-preserving pairs. For each training batch, we compute pairwise WL scores and construct:

- Positive pairs: (G_i, G_j) where $\text{sim}_{\text{WL}}(G_i, G_j) > \tau_{\text{high}}$,
- Negative pairs: (G_i, G_j) where $\text{sim}_{\text{WL}}(G_i, G_j) < \tau_{\text{low}}$.

A Graph Isomorphism Network (GIN) [17] encoder GIN_{hw} produces embeddings $\mathbf{h}^i_{\text{hw}} = \text{GIN}_{\text{hw}}(G_i)$ trained with contrastive loss:

$$\mathcal{L}_{\text{hier}} = -\log \frac{\sum_{j \in \mathcal{P}(i)} \exp(\mathbf{h}^i_{\text{hw}} \cdot \mathbf{h}^j_{\text{hw}} / \tau)}{\sum_{k \neq i} \exp(\mathbf{h}^i_{\text{hw}} \cdot \mathbf{h}^k_{\text{hw}} / \tau)}, \tag{5}$$

where $\mathcal{P}(i)$ denotes positive samples for anchor i and τ is temperature. This approach captures structural invariances correlating with power behavior without requiring expensive RTL simulation.

4.3 Multi-Task Training with Power Metrics

Feature Aggregation. We concatenate embeddings from all compilation stages—$\mathbf{h}_{\text{sw}}$ from software compilation and $\mathbf{h}_{\text{hw}}$ from hardware elaboration (when applicable)—into unified features $\mathbf{h}$ capturing coupled software-hardware semantics. Given $\mathbf{h}$, we formulate multi-task regression jointly predicting average power P_{avg} and total energy E_{total}:

$$\hat{P}_{\text{avg}} = f_P(\mathbf{h}; \theta_P), \quad \hat{E}_{\text{total}} = f_E(\mathbf{h}; \theta_E), \tag{6}$$

where f_P and f_E are task-specific heads sharing a common trunk of fully-connected layers, θ_P and θ_E are the parameters of corresponding prediction head.

Training Objective and Mutual Reinforcement. The multi-task loss is defined as:

$$\mathcal{L}_{\text{total}} = \mathcal{L}_{\text{Power}} + \lambda \cdot \mathcal{L}_{\text{Energy}} = \text{MSE}(\hat{P}_{\text{avg}}, P_{\text{avg}}) + \lambda \cdot \text{MSE}(\hat{E}_{\text{total}}, E_{\text{total}}), \tag{7}$$

where λ is a task weighting coefficient. Joint training provides complementary supervision: energy constraints penalize inconsistent power predictions (e.g., high power but low energy), while power patterns distinguish workloads with similar energy but different temporal profiles. This mutual reinforcement acts as an implicit regularization [18], critical for generalization with limited data.

Two-stage Training Strategy. As illustrated in Figure 3, our framework decouples data requirements across stages. *Stage 1* trains compilation encoders using self-supervised objectives ($\mathcal{L}_{\text{sw}}$, $\mathcal{L}_{\text{hw}}$) with abundant, inexpensive compilation artifacts—requiring no RTL simulation. *Stage 2* freezes pre-trained encoders and trains prediction heads (f_P, f_E) using limited post-synthesis power data from synthesis and simulation. Multi-task learning amplifies sample efficiency: each expensive simulation provides dual supervision signals whose mutual reinforcement through $\mathcal{L}_{\text{total}}$ reduces required samples.

5 Experiment

To validate the generality of our compilation-based power prediction methodology across diverse RTL generator architectures, we conduct comprehensive experiments on two representative platforms: Gemmini [1] and hls4ml [2]. Gemmini exemplifies Chisel-based RTL generators that employ dual compilation chains—a software compiler translating neural networks into accelerator-specific instruction binaries and a hardware compiler elaborating parameterized Chisel descriptions through FIRRTL intermediate representations to synthesizable RTL. In contrast, hls4ml represents HLS-based generators that leverage unified MLIR-based progressive lowering, transforming high-level neural network specifications through multiple abstraction levels into hardware implementations. These two cases collectively cover the predominant agile development methodologies for DNN accelerators in current practice.

5.1 Case 1: Gemmini Accelerator

Expeimental Setup. We construct our dataset using the Chipyard [22] framework, encompassing 26 configurable Gemmini SoC parameters spanning systolic array, memory hierarchy, dataflow strategies, etc. The workload suite includes variants of neural network architectures—MLPs, CNNs, ResNets [23], MobileNet [24], Transformers-small [15], and BERT-base [25]—with model complexity ranging up to 100 million parameters. For training the feature extractors, we leverage over 20,000 pre-compilation and post-compilation data pairs generated from diverse hardware-network combinations. For each hardware-network configuration pair, we obtain ground-truth power measurements through RTL simulation using Synopsys VCS, gate-level synthesis with Cadence Genus targeting the ASAP7 [26] 7nm process, and power analysis using Cadence Joules. This yields 6,021 (hardware configuration, network workload, power profile) data triples. Both average total power (mW) and total energy consumption (mJ) are selected as target prediction metrics.

To comprehensively assess generalization capability, we evaluate under three partitioning strategies:

- *Random Split*—standard 80/20 train-test division measuring baseline accuracy under matched distributions;
- *Scale Generalization*—training on small networks while testing on large models to verify extrapolation of learned power-composition relationships;
- *Architecture Generalization*—training on MLP and CNN variants while testing on ResNet and MobileNet variants(Arch_1), or testing on Transformer and Bert variants(Arch_2).

We report mean absolute percentage error (MAPE) quantifying relative accuracy across consumption magnitudes, and coefficient

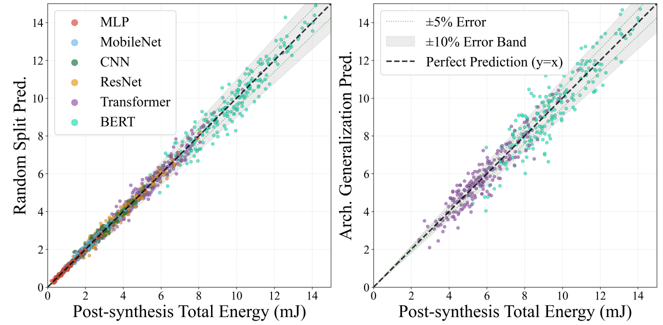

Figure 4: Our Method's Prediction v.s. Ground Truth.

of determination (R^2) measuring explained variance. For simulator-based methods, we report Kendall's τ coefficient to measure ranking correlation for fairness. These metrics jointly characterize point-wise prediction fidelity and correlation strength across the design space.

Comparison against Previous Methods. The agile design scenario demands power prediction methodologies with two critical characteristics: rapid iteration capability and minimal expert dependency. To the best of our knowledge, this work represents the first power prediction framework on DNN accelerators with these requirements. Consequently, baseline selection focuses on methodological paradigms rather than directly comparable prior art. We select baselines representing three distinct modeling methodologies:

(1) Simulator-based methods. Timeloop/Accelergy [4, 5] represents the mainstream architecture-level DNN energy simulation approach. We employ Gemmini's officially provided template, though its coverage of configurable parameters is limited. Since architecture-level abstractions inherently deviate from ground truth power, we adopt Kendall's τ coefficient for fair comparison despite this systematic bias.

(2) Kernel-aware prediction. The nn-Meter [19] originally targets mobile devices' latency prediction through kernel-level analysis of compilation artifacts under standard test inputs. We adapt its methodology to the RTL generator context by extracting features from Gemmini's compiled computational graphs.

(3) Surrogate model with manual feature engineering. DANCE [20] and QUIDAM [21] represent approaches that construct surrogate models using explicit hardware parameter vectors combined with hand-crafted neural network features to characterize workload.

Table 1 presents the quantitative comparison results. Our approach demonstrates superior accuracy across all evaluation scenarios. Under random split, our method achieves 6.34% MAPE for power prediction with $R^2 = 0.975$, and 4.95% MAPE for energy prediction with $R^2 = 0.983$. The prediction quality remains robust under challenging generalization scenarios: scale generalization yields 7.20% MAPE for power and 6.68% for energy, while architecture generalization on ResNet/MobileNet variants (Arch_1) achieves 8.30% and 7.27% MAPE, respectively. As shown in Figure 4, even for the most challenging Transformer/BERT generalization (Arch_2), our method maintains reasonable accuracy at 11.03% and 9.73% MAPE.

The accuracy gap with simulator-based methods reflects a fundamental limitation: while Gemmini exposes over 100 configurable parameters, architecture-level simulators like Timeloop can practically model only 10-20 critical ones, establishing an inherent accuracy ceiling. Our compilation-driven approach directly processes RTL artifacts reflecting all configured parameters. Notably, compilation-based methods (nn-Meter and ours) outperform feature engineering approaches across all scenarios. This validates our core insight that

Table 1: Comparison of Power and Energy Prediction against Current Methods on Gemmini.

Data Split	Metric	Accelergy [5]	nn-Meter [19]		DANCE [20]		QUIDAM [21]		Ours		
		Kendall's τ	MAPE(%)	R^2	MAPE(%)	R^2	MAPE(%)	R^2	Kendall's τ	MAPE(%)	R^2
Random	Power	0.508	18.94	0.843	23.39	0.736	25.39	0.721	**0.928**	**6.34**	**0.975**
	Energy	0.562	15.90	0.872	22.30	0.747	24.02	0.730	**0.939**	**4.95**	**0.983**
Scale	Power	0.527	21.44	0.749	28.47	0.690	25.90	0.717	**0.907**	**7.20**	**0.963**
	Energy	0.618	20.56	0.757	26.93	0.712	24.44	0.725	**0.920**	**6.68**	**0.972**
Arch_1	Power	0.428	28.46	0.692	N/A	N/A	N/A	N/A	**0.894**	**8.30**	**0.953**
	Energy	0.500	27.35	0.708	N/A	N/A	N/A	N/A	**0.910**	**7.27**	**0.964**
Arch_2	Power	0.619	31.95	0.570	N/A	N/A	N/A	N/A	**0.863**	**11.03**	**0.921**
	Energy	0.637	29.04	0.623	N/A	N/A	N/A	N/A	**0.886**	**9.73**	**0.946**

N/A denotes not available (unsupported by the method).

Table 2: Comparison of Power and Energy Prediction against Current Methods on hls4ml.

Data Split	Metric	rule4ml [27]		wa-hls4ml [28]		Ours	
		MAPE(%)	R^2	MAPE(%)	R^2	MAPE(%)	R^2
Random	Power	25.82	0.703	20.34	0.812	**7.92**	**0.957**
	Energy	23.16	0.739	19.20	0.830	**6.59**	**0.970**
Scale	Power	32.94	0.539	28.49	0.603	**8.34**	**0.945**
	Energy	29.53	0.598	27.01	0.688	**8.01**	**0.953**
Arch_1	Power	N/A	N/A	N/A	N/A	**12.93**	**0.892**
	Energy	N/A	N/A	N/A	N/A	**12.11**	**0.901**

N/A denotes not available (unsupported by the method).

compiler IRs provide superior abstraction for unified feature extraction from diverse neural network architectures.

The average runtime of our framework takes about 15 seconds per query, from user input (NN and HW configs) through compilation to final power prediction (even faster with batch input), laying a solid foundation for further applications in agile design flows, e.g. design space exploration, dnn-hardware co-design, etc.

5.2 Case 2: hls4ml

Expeimental Setup. For the hls4ml framework, we leverage the Xilinx Vitis HLS toolchain to construct our experimental dataset. Unlike hardware-parameterized accelerators, hls4ml transforms neural networks directly from software-level model specifications without explicit hardware configuration parameters. The neural network workload suite comprises MLPs, CNNs, ResNet, RNNs, and MobileNet variants, with model complexity scaling up to 30 million parameters. For each network workload, we perform high-level synthesis with default optimization strategies and fix *ReuseFactor* to 1. We set $ap_fixed < 8, 3 >$ for every layer via the hls4ml configuration. Power and energy metrics are extracted through post-synthesis power analysis. This process generates 2,879 experimental data pairs consisting of (network configuration, power/energy report). For training the feature extractor, we leverage 5,364 pre-compilation and post-compilation data pairs generated from diverse network configurations. Consistent with the Gemmini experiments, we select both average power consumption (mW) and total energy (mJ) as our target prediction metrics.

Similar to Section 5.1, we evaluate under three partitioning strategies: Random Split, Scale Generalization, and Architecture Generalization, with MAPE and R^2 as metrics.

Comparison against Previous Methods. We compare our approach against two state-of-the-art power modeling methods for hls4ml: rule4ml [27] and wa-hls4ml [28]. Both baselines rely on manual feature engineering to characterize input neural network models. Rule4ml extracts hand-crafted features from network architectures and represents them as fixed-length vectors, which are then fed into regression models for power prediction. Wa-mls4ml employs a graph-based representation, manually constructing graphs from network specifications and utilizing graph neural networks for prediction. Despite their effectiveness on seen architectures, both methods face a critical limitation: they lack generalization capability to predict power consumption for neural network architectures absent from the training set.

Table 2 demonstrates advantages in the hls4ml context. Our approach achieves 7.92% MAPE for power and 6.59% for energy under random split, with R^2 exceeding 0.95. Scale generalization yields 8.34% and 8.01% MAPE, while architecture generalization maintains 12.93% and 12.11% MAPE. Baseline methods are functionally not available for architecture generalization due to hand-crafted features' inability to handle network architectures absent from training, which further underscores the value of our unified compilation-driven representation for robust and practically deployable prediction across evolving network families.

6 Conclusion

This work introduces a paradigm shift in power modeling for DNN accelerator agile design: *compilation tells energy*. We recognize that RTL generators' compilation processes already fuse workload characteristics with hardware constraints—the key coupling determining power. By extracting features directly from compiler intermediate representations, our methodology naturally aligns with agile development workflows: it requires no architecture simulation expertise, executes through push-button automated compilation, and provides rapid power feedback in seconds. Through two-stage learning that decouples abundant compilation data from limited RTL simulation, combined with multi-task training, we achieve practical accuracy. Validation on Gemmini and hls4ml demonstrates broad applicability across diverse generator architectures, enabling high prediction accuracy in agile design scenarios.

Looking forward, the rapid, push-button power feedback enables algorithm engineers to evaluate deployment costs during early-stage model design, while facilitating efficient design space exploration for automatic accelerator design. This establishes critical infrastructure where power-aware decisions—from neural architecture choices to accelerator configurations—can be made within seconds rather than hours, truly enabling co-design throughout the development cycle in a practical, iteration-friendly manner for agile teams working under tight development schedules and repeated design-turnaround constraints.

Acknowledgment

This work is sponsored by Natural Science Foundation of Shanghai (Project No.25JD1403000) and the National Natural Science Foundation of China (No. 62404021).

References

[1] H. Genc *et al.*, "Gemmini: Enabling systematic deep-learning architecture evaluation via full-stack integration," in *Proc. DAC*, 2021.
[2] F. Fahim, B. Hawks, C. Herwig, J. Hirschauer, S. Jindariani, N. Tran, L. P. Carloni, G. Di Guglielmo, P. Harris, J. Krupa *et al.*, "hls4ml: An open-source codesign workflow to empower scientific low-power machine learning devices," *arXiv preprint arXiv:2103.05579*, 2021.
[3] 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.
[4] A. Parashar, P. Raina, Y. S. Shao, Y.-H. Chen, V. A. Ying, A. Mukkara, R. Venkatesan, B. Khailany, S. W. Keckler, and J. Emer, "Timeloop: A systematic approach to dnn accelerator evaluation," in *2019 IEEE international symposium on performance analysis of systems and software (ISPASS)*. IEEE, 2019, pp. 304–315.
[5] Y. N. Wu, J. S. Emer, and V. Sze, "Accelergy: An architecture-level energy estimation methodology for accelerator designs," in *2019 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. IEEE, 2019, pp. 1–8.
[6] 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.
[7] J. Zhai, C. Bai, B. Zhu, Y. Cai, Q. Zhou, and B. Yu, "McPAT-Calib: A microarchitecture power modeling framework for modern CPUs," in *Proc. ICCAD*. IEEE, 2021, pp. 1–9.
[8] J. Zhai, Y. Cai, and B. Yu, "Microarchitecture Power Modeling via Artificial Neural Network and Transfer Learning," in *Proc. ASPDAC*, 2023, pp. 1–6.
[9] Q. Zhang, S. Li, G. Zhou, J. Pan, C.-C. Chang, Y. Chen, and Z. Xie, "PANDA: Architecture-level power evaluation by unifying analytical and machine learning solutions," in *Proc. ICCAD*. IEEE, 2023, pp. 01–09.
[10] Q. Zhang, Y. Lu, M. Li, and Z. Xie, "Autopower: Automated few-shot architecture-level power modeling by power group decoupling," in *Proc. DAC*. IEEE, 2025, pp. 1–7.
[11] 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*, 2009, pp. 469–480.
[12] A. Izraelevitz, J. Koenig, P. Li, R. Lin, A. Wang, A. Magyar, D. Kim, C. Schmidt, C. Markley, J. Lawson *et al.*, "Reusability is FIRRTL ground: Hardware construction languages, compiler frameworks, and transformations," in *2017 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. IEEE, 2017, pp. 209–216.
[13] C. Lattner, M. Amini, U. Bondhugula, A. Cohen, A. Davis, J. Pienaar, R. Riddle, T. Shpeisman, N. Vasilache, and O. Zinenko, "MLIR: Scaling compiler infrastructure for domain specific computation," in *2021 IEEE/ACM International Symposium on Code Generation and Optimization (CGO)*. IEEE, 2021, pp. 2–14.
[14] T. Kipf, "Semi-supervised classification with graph convolutional networks," *arXiv preprint arXiv:1609.02907*, 2016.
[15] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," *Advances in neural information processing systems*, vol. 30, 2017.
[16] D. Luo, Q. Sun, X. Li, C. Zhuo, B. Yu, and H. Geng, "From Flatland to Forest: Exploring Pareto-optimal Design through RTL Hierarchy Trees," in *2025 62nd ACM/IEEE Design Automation Conference (DAC)*. IEEE, 2025, pp. 1–7.
[17] K. Xu, W. Hu, J. Leskovec, and S. Jegelka, "How powerful are graph neural networks?" *arXiv preprint arXiv:1810.00826*, 2018.
[18] S. Ruder, "An overview of multi-task learning in deep neural networks," *arXiv preprint arXiv:1706.05098*, 2017.
[19] L. L. Zhang, S. Han, J. Wei, N. Zheng, T. Cao, Y. Yang, and Y. Liu, "nn-meter: Towards accurate latency prediction of deep-learning model inference on diverse edge devices," in *Proceedings of the 19th Annual International Conference on Mobile Systems, Applications, and Services*, 2021, pp. 81–93.
[20] K. Choi, D. Hong, H. Yoon, J. Yu, Y. Kim, and J. Lee, "DANCE: Differentiable accelerator/network co-exploration," in *2021 58th ACM/IEEE Design Automation Conference (DAC)*. IEEE, 2021, pp. 337–342.
[21] "QUIDAM: A Framework for Quantization-aware DNN Accelerator and Model Co-Exploration, author=Inci, Ahmet and Virupaksha, Siri and Jain, Aman and Chin, Ting-Wu and Thallam, Venkata and Ding, Ruizhou and Marculescu, Diana," *ACM Transactions on Embedded Computing Systems*, vol. 22, no. 2, pp. 1–21, 2023.
[22] A. Amid *et al.*, "Chipyard: Integrated design, simulation, and implementation framework for custom socs," *IEEE Micro*, 2020.
[23] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.
[24] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, "Mobilenets: Efficient convolutional neural networks for mobile vision applications," *arXiv preprint arXiv:1704.04861*, 2017.
[25] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of deep bidirectional transformers for language understanding," *arXiv preprint arXiv:1810.04805*, 2018.
[26] L. T. Clark, V. Vashishtha, L. Shifren, A. Gujja, S. Sinha, B. Cline, C. Ramamurthy, and G. Yeric, "ASAP7: A 7-nm finFET predictive process design kit," *Microelectronics Journal*, vol. 53, pp. 105–115, 2016.
[27] M. M. Rahimifar, H. E. Rahali, and A. C. Therrien, "rule4ml: an open-source tool for resource utilization and latency estimation for ML models on FPGA," *Machine Learning: Science and Technology*, vol. 6, no. 1, p. 015009, 2025.
[28] B. Hawks, D. Plotnikov, N. Tran, K. Tame-Narvaez, M. M. Rahimifar, H. E. Rahali, A. C. Therrien, G. Di Guglielmo, J. Duarte, and V. Loncar, "wa-hls4ml and lui-gnn: A Benchmark and GNN based Surrogate Model for hls4ml Resource and Latency Estimation," in *Proceedings of the 2025 ACM/SIGDA International Symposium on Field Programmable Gate Arrays*, 2025, pp. 43–43.