Improving Energy Efficiency of Machine Learning Software with an Instruction-Level Dynamic Energy Model for DNN Accelerators

Jonathan Wang

Electrical Engineering and Computer Sciences
University of California, Berkeley

Technical Report No. UCB/EECS-2025-124
http://www2.eecs.berkeley.edu/Pubs/TechRpts/2025/EECS-2025-124.html

May 19, 2025

Improving Energy Efficiency of Machine Learning Software with an Instruction-Level Dynamic Energy Model for DNN Accelerators

by Jonathan Wang

Research Project

Submitted to the Department of Electrical Engineering and Computer Sciences, University of California at Berkeley, in partial satisfaction of the requirements for the degree of **Master of Science, Plan II**.

Approval for the Report and Comprehensive Examination:

Committee:

Yakun Sophia Shao

Professor Yakun Sophia Shao
Research Advisor

5/19/2025

(Date)

* * * * * * *

Professor Borivoje Nikolic
Second Reader

5/19/2025

(Date)

Improving Energy Efficiency of Machine Learning Software with an Instruction-Level Dynamic Energy Model for DNN Accelerators

by
Jonathan Franklin Wang

A thesis submitted in partial satisfaction of the
requirements for the degree of
Master of Science
in
Electrical Engineering and Computer Sciences
in the
Graduate Division
of the
University of California, Berkeley

Committee in Charge:

Professor Yakun Sophia Shao, Chair
Professor Borivoje Nikolic

Spring 2025

Abstract

Improving Energy Efficiency of Machine Learning Software with an Instruction-Level Dynamic Energy Model for DNN Accelerators

by Jonathan Franklin Wang

Master of Science in Electrical Engineering and Computer Sciences

University of California, Berkeley

Professor Yakun Sophia Shao, Chair

Energy-efficient software development on domain-specific processors necessitates a high-level model to quickly and accurately estimate software workload energy consumption. However, previous work in energy characterization has mostly focused on architectural design space exploration for domain-specific processors or ISA-based energy models for CPUs. Currently, no work has been done to create ISA-based energy models targeting DNN accelerators.

In this thesis, we present a methodology for creating instruction-level energy models specific to DNN accelerators, focusing on architectures that use systolic arrays for computation. For energy characterization, we determine the DNN accelerator's energy consumption for both its systolic array and private SRAMs for each accelerator instruction. Based on this characterization, we estimate the energy cost per accelerator instruction, which is used to predict the overall energy consumption of DNN accelerator workloads. By using this methodology, our energy models make predictions that are 90% accurate to the actual energy consumption of real-world benchmarks.

Contents

List of Figures

List of Tables

Acknowledgements

First, I want to thank my advisor, Professor Sophia Shao, for showing me the fascinating world that is hardware for machine learning and for guiding me when I explored different paths for my research projects. I also want to thank Professor Borivoje "Bora" Nikolic for reviewing this thesis and guiding me through the tape-out and bring-up processes throughout my studies.

Furthermore, I owe a deep gratitude to Charles Hong and Nayiri Krzysztofowicz, both of whom helped me understand and set up the infrastructure for Timeloop, Chipyard, and the power/energy prediction flow. Without the guidance of either, I would not have been able to write this thesis. Lastly, I would like to thank all the undergraduate and graduate students from the SLICE and BWRC labs for making the research experience that much more enjoyable.

Chapter 1

Introduction

Artificial intelligence has seen a significant surge of interest in the past decade, resulting in many breakthroughs [9–13] that introduced fundamental paradigm shifts in the way machine learning software is written. With the slowdown of technology scaling, modern system-on-chip (SoC) designs have increasingly relied on incorporating deep neural network (DNN) accelerators [14–17] in order to improve the energy efficiency of machine learning training and inference.

DNN accelerators improve energy efficiency by maximizing data parallelism and data reuse for computationally heavy machine learning software. DNN accelerators differ from general-purpose processors because they have spatial architectures as opposed to temporal architectures. Consequently, the programming interface and instruction set architecture (ISA) for DNN accelerators can be quite different compared to CPUs.

Significant focus has been put towards maximizing theoretical power, performance, and area (PPA) of DNN accelerator hardware. However, real software workloads are often unable to achieve the highest theoretical energy efficiency due to lower utilization and denser matrices. Although achieving a DNN accelerator's maximum theoretical energy efficiency is not possible for every workload, there remain opportunities to optimize. For a given workload, software strategies for tiling, dataflow, and spatio-temporal mapping can greatly impact its energy consumption. These software strategies do not change what the workload outputs, but rather how it is executed on the hardware.

To solve problems in this space, Design Space Exploration (DSE) algorithms are created to optimize software strategies for specific DNN accelerator hardware, though they are more suited to aid architects than developers. Much less work exists to help developers optimize the energy efficiency of hand-designed code built to run on this hardware. From a developer's perspective, energy efficiency is most interpretable when their compiled, executable code is given a numerical energy

consumption value. In this environment, developers can constantly iterate through their code in order to maximize energy efficiency without being required to understand every microarchitectural detail behind their host DNN accelerator.

To improve software development for DNN accelerators, this work contributes a methodology to create a developer-friendly model for energy evaluations of DNN accelerator workloads. Specifically:

- We determine the energy per instruction (EPI) for each unique instruction in a DNN accelerator's ISA using RTL-level energy estimations from commercial tools.
- We integrate the EPI estimates into a functional simulator to construct an instruction-level model that utilizes EPI to determine the overall energy consumption for software workloads.
- We evaluate our methodology using the Gemmini accelerator, creating a model that predicts workload energy consumption with an accuracy of 90% in a fraction of the time compared to commercial tools for energy estimation.

Chapter 2

Background

DNN accelerators have unique architectures that differ from general-purpose processors, resulting in new challenges that must be faced when designing microarchitectures for them. Thus, DNN accelerator energy models must be able to capture microarchitectural impacts among designs to become more accurate and useful to architects and software developers alike. To meet this need, an instruction-level energy model for DNN accelerators is necessary to guide energy efficiency improvements to software workloads.

2.1 Domain-Specific Processors

The end of Moore's Law has prompted both industry and academia to recognize that improved SoC performance year after year is no longer possible by solely increasing the clock frequency in general-purpose CPUs. Instead, architects have designed hardware accelerators to continue performance gains across specialized domains, including, but not limited to, graphics, robotics, digital signal processing, and deep learning. These domain-specific accelerators have significantly higher energy efficiency than general-purpose CPUs, yet continued optimizations to these architectures are necessary to continue yielding higher performance year after year.

2.1.1 Deep Neural Network Accelerators

DNN accelerators are domain-specific accelerators that focus on improving the performance and energy efficiency of machine learning workloads. Since machine learning software is comprised of a significant amount of matrix and vector operations, DNN accelerators utilize spatial architectures

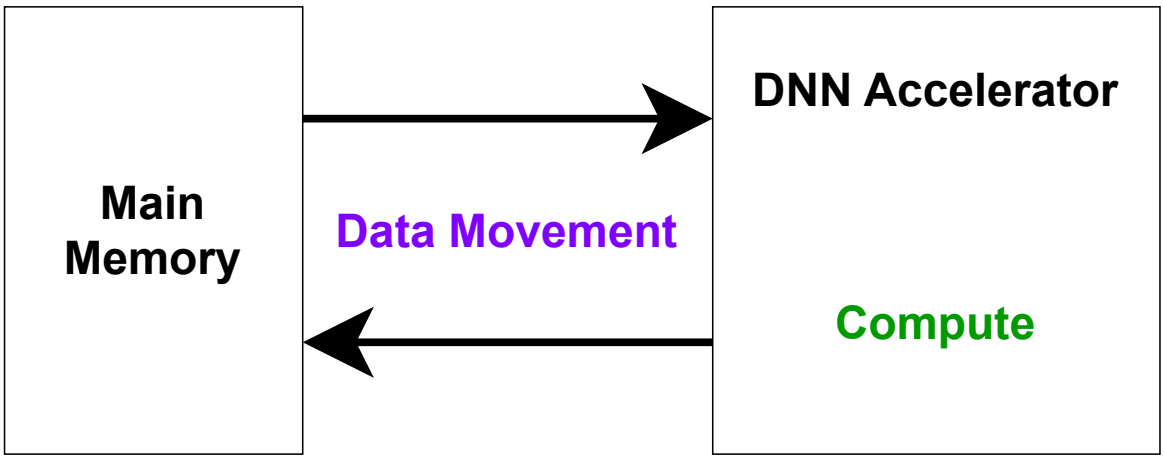


FIGURE 2.1: Data movement involves sending data between main memory and the accelerator. Between data entering and leaving the accelerator, computation occurs inside the accelerator.

to reduce the compute bottleneck that plagues temporal architectures. However, DNN accelerators face new challenges in the areas of data movement and computation.

Data movement involves sending data from the SoC main memory to the accelerator and vice versa. Data movement is a challenging problem because machine learning model parameters—input activations, weights, and output activations—take up a significant memory footprint and cannot all be stored in accelerator memory simultaneously. Instead, only portions of the model parameters can be stored and processed by the accelerator at a given time. Thus, reducing memory transaction requests and their associated latency becomes important to maximizing performance.

Once data is loaded into the accelerator, it performs arithmetic-heavy computations. Since DNN accelerators use spatial architectures, multiply-accumulate (MAC) operations can be heavily parallelized to improve throughput and reduce overall workload latency. Computation is a challenging problem because large matrix operations must be split into tiles in order to be processed by the accelerator, where the tile size depends on the number of hardware MAC units provided by the accelerator. Full matrix operations are completed over multiple DNN accelerator instructions, depending on the tile size. As a result, data reuse and MAC utilization strategies serve as crucial design points for DNN accelerators.

Although there are many different DNN accelerator architectures, some hardware blocks are shared between designs because of their effectiveness in handling data movement and computation challenges. Many DNN accelerators have their own private memory, comprised of SRAM or DRAM blocks, to speed up data access time compared to the SoC's main memory. For processing matrix operations, systolic arrays are popular for matrix multiplication and convolution operations because they can use their high MAC parallelization and utilization to speed up machine learning software.

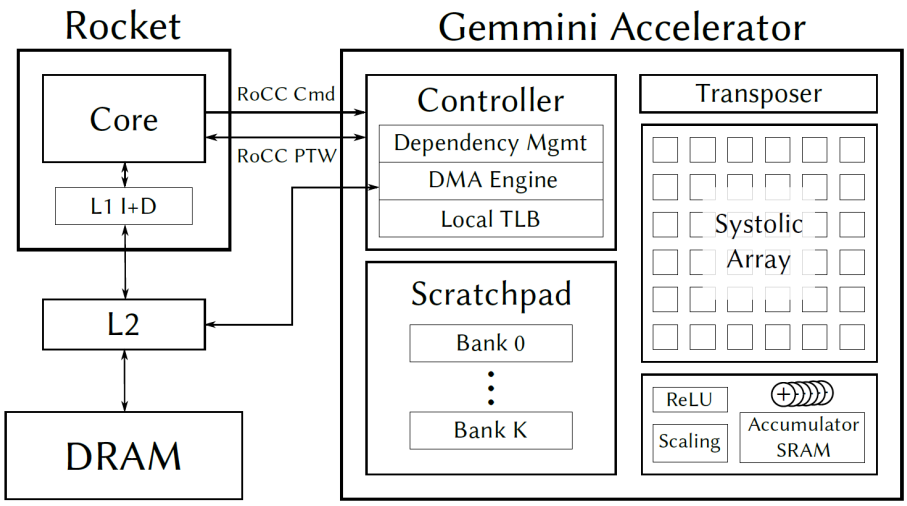


FIGURE 2.2: Architectural block diagram of Gemmini [1].

2.1.2 Gemmini

Gemmini [1] is a platform for DNN accelerator systems, comprised of a Chisel-based [18] DNN accelerator generator and a mature software stack. Gemmini produces synthesizable RTL for DNN accelerators that can be integrated into a larger SoC. Gemmini is configurable, enabling the generation of DNN accelerators with different datatypes, in-accelerator memory sizes, and systolic array dimensions. The software stack can run on any created Gemmini hardware configuration.

Gemmini is part of the larger Chipyard [6] ecosystem that provides a comprehensive platform for generating and simulating complete SoC designs. Gemmini can be integrated into Chipyard SoCs as a tightly-coupled accelerator to a host RISC-V CPU. Gemmini's programming interface includes custom instructions that serve as an extension to the RISC-V ISA, with instructions for configuration, data movement, and computation (see Table 2.1 for a list of instructions).

Gemmini's private memory system, which includes the scratchpad and accumulator SRAMs, is row-addressed with 32-bit addresses. Each row contains DIM elements, where DIM is a parameter for Gemmini's hardware generator that is used to determine the systolic array's dimensions. Both the scratchpad and the accumulator have DIM elements per row, but the number of rows and the element's datatype can differ between these modules. Figure 2.3 shows how individual rows of memory can be accessed with the row-addressing system. For this work, we characterize the dynamic energy of Gemmini with a 16×16 systolic array, int8 scratchpad, and an int32 accumulator.

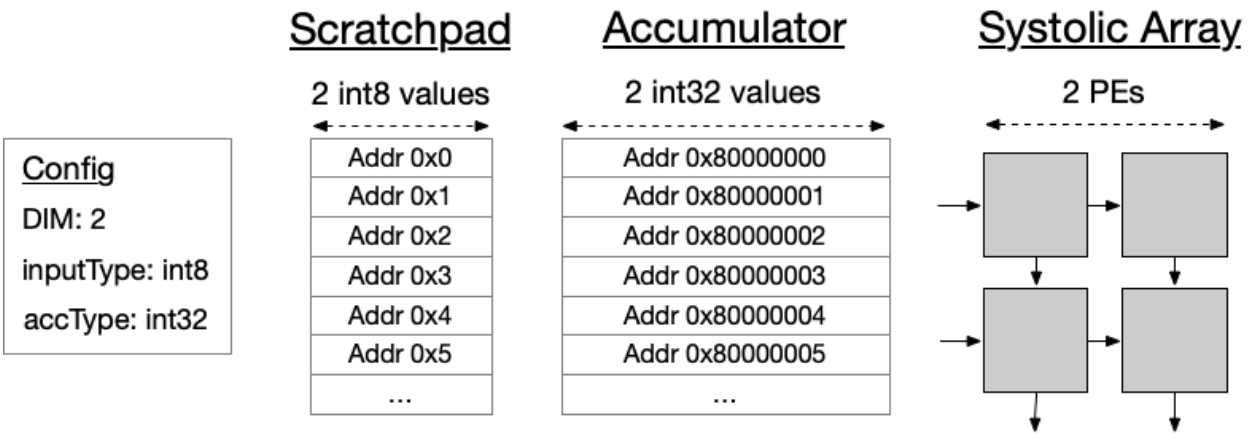


FIGURE 2.3: Gemmini private memory internals with example config.

TABLE 2.1: Gemmini's ISA with descriptions of each instruction in the WS dataflow.

Instruction	Instruction Type	Description
config_ld	Configuration	Set stride and scaling factor for mvin instructions
config_st	Configuration	Set stride for mvout instructions
config_ex	Configuration	Set dataflow, stride, scaling factor, and activation function for compute instructions
mvin	Data Movement	Move up to DIM rows and 4 × DIM columns of matrix data from main memory → accelerator memory
mvout	Data Movement	Move up to DIM rows and 4 × DIM columns of matrix data from accelerator memory → main memory
preload	Compute	Set C's output address in accelerator memory and load B into the systolic array
compute_preloaded	Compute	Load A into the systolic array then compute a matrix multiplication with the new B
compute_accumulated	Compute	Load A into the systolic array then compute a matrix multiplication with the previous B

Gemmini accelerates general matrix multiply (GEMM) operations, with its ISA enabling substantial configuration for how they are executed. For machine learning workloads, GEMM represents A as input activations, B as weights, D as biases, and C as output activations. Mathematically, GEMM is defined as:

$$C = A \times B + D$$

There are many ways to compute GEMM operations, but dataflow stands out as an important factor because it is one of Gemmini's programmable features. Dataflow drives the arithmetic operation order, affecting which matrix values are reused, when they are reused, and how often they are reused. For Gemmini, specifically, configuring the dataflow modifies the way instructions are executed on its hardware. There are multiple GEMM dataflows, but Gemmini only supports two: Output-stationary (OS) and weight-stationary (WS). For this work, our energy model only considers the WS dataflow for energy characterization of Gemmini instructions.

```
int8_t A[DIM_I][DIM_K];
int8_t B[DIM_K][DIM_J];
int32_t C[DIM_I][DIM_J];

for(size_t k = 0; k < DIM_K; k++) {
    for(size_t j = 0; j < DIM_J; j++) {
        for(size_t i = 0; i < DIM_I; i++) {
            C[i][j] += A[i][k] * B[k][j];
        }
    }
}
```

Listing 2.1: Weight-stationary matrix multiplication.

When using the WS dataflow, B data reuse is prioritized over D data reuse. Gemmini configured to the WS dataflow specifies that the preload instructions load B into the systolic array while the compute instructions feed A and D into the systolic array to compute C, which is then stored in the accumulator. A and B are stored in the scratchpad, while D is stored in the accumulator.

2.2 Chip-Level Power and Energy

Chip-level power consumption can be analyzed through the numerous CMOS transistors that make up the standard cells and memory block macros of a SoC. All power consumption on an SoC is considered to be static or dynamic power.

Static power, also known as leakage power, is caused by the leakage current flowing through standard cell transistors because transistor gates are non-ideal and possess additional capacitance. Regardless of whether the transistors are off, the leakage current can still flow. As a result, static power is highly dependent on process node technology, CMOS transistor count, and clock frequency. Static power remains even when logic gates are not being used, so memory blocks—dense in transistor count and usually not fully utilized—tend to contribute more static than dynamic power.

Dynamic power comes from two different sources, switching power and short-circuit power. Switching power increases the more times a logic gate switches its output, either from $1 \rightarrow 0$ or $0 \rightarrow 1$, due to the charging or discharging of the load capacitance for every switch. Short-circuit power occurs because there is a brief moment when the pFET and nFET transistors are on during switching, which causes a short-circuit current to flow through the logic gate. Dynamic power is heavily dependent on the software workload, especially with respect to instruction counts and data sparsity.

2.2.1 Static Energy

Static power remains constant, regardless of the workload running on the hardware. Thus, the static energy scales linearly with the workload latency. Since hardware has a greater impact on static energy than software, this work does not consider static energy for energy efficiency. Instead, this work focuses on the energy impact of different software workloads running on identical hardware.

2.2.2 Dynamic Energy

$$\mathrm{P_{dynamic}} = \mathrm{P_{short\text{-}circuit}} + \mathrm{P_{switching}}\ \mathrm{mW}$$

$$\mathrm{E_{dynamic}} = \mathrm{P_{dynamic}} \times 10^3 \times \frac{\mathrm{cycles}}{\mathrm{frequency}}\ \mathrm{uJ}$$

Dynamic energy for a specific workload is computed by summing up short-circuit and switching power, then multiplying the combined dynamic power with the workload's latency, determined using the cycle count and clock frequency. In this work, the dynamic energy of a workload is used to determine its energy efficiency. Dynamic energy is a useful evaluation metric because it accounts for both workload power consumption and latency.

2.3 Prior Work

Prior work for DNN accelerator energy modeling has been done mainly at the architectural stage through fast analytical models with less than ideal accuracy. However, work has also been done for more general-purpose energy models, which provide frameworks and methodologies applicable to DNN accelerator energy modeling. An opportunity lies in constructing an energy model based on RTL-level over architecture-level dynamic energy to improve model prediction accuracy. Overall, there is a need for DNN accelerator energy models to comprehend microarchitectural design choices while maintaining low-latency evaluation speeds.

2.3.1 DNN Accelerator Energy Models

Existing energy models for DNN accelerators have emphasized their use for DSE at the architectural level. The Deep Neural Network Energy Estimation Tool [19] is an analytical model that predicts the dynamic energy consumption per layer of a DNN workload using the DNN model's layer dimensions and the number of bits accessed per memory hierarchy level as input. Although great for comparing dynamic energy consumption between different DNN models, this tool does not account for energy consumption variation between different software implementations of the same model.

Timeloop [20] is a more robust analytical model designed to find the most optimal hardware configuration and software implementation for a target DNN model. Timeloop uses Accelergy's [21] energy model by utilizing dynamic energy cost per access of key hardware components to predict workload dynamic energy. These energy predictions are used as input to a mapper to select the most energy-efficient and valid algorithm-to-hardware mapping for workloads on DNN accelerators. DOSA [22] is an extension of Timeloop that automates DSE using a similar access-based model to find optimal hardware configurations and algorithm-to-hardware mappings for DNN workloads.

Although these analytical models provide fast evaluations that are useful for continuous DSE iteration, they are unable to consider the microarchitectural impacts to dynamic energy consumption. As a result, their accuracy is significantly reduced compared to the energy consumption at signoff. DNN accelerator energy modeling can benefit from an energy model that considers microarchitectural design choices.

2.3.2 CPU Instruction-Level Energy Models

In the field of CPU energy modeling, there have been promising methods to increase prediction accuracy while maintaining fast prediction times, namely instruction-level energy models. These energy models estimate the EPI for all instructions in the CPU's ISA and, depending on how each instruction is used, can be calculated for the same instruction in various architectural states to capture possible hardware or software scenarios that need to be considered by the energy model. Work has been done to create an instruction-level dynamic energy model for Intel's high-performance computing Xeon Phi processor [23]. Through computing EPI for CPU instructions with different changes to the processor—such as number of cores, threads, and operands—this energy model has achieved an accuracy between 1% and 5% on real-world benchmarks. Additionally, work on an instruction-level dynamic energy model for the RISC-V ISA, using the Rocket CPU [24], has quantitatively shown the energy impact of different arithmetic and memory access instructions [25].

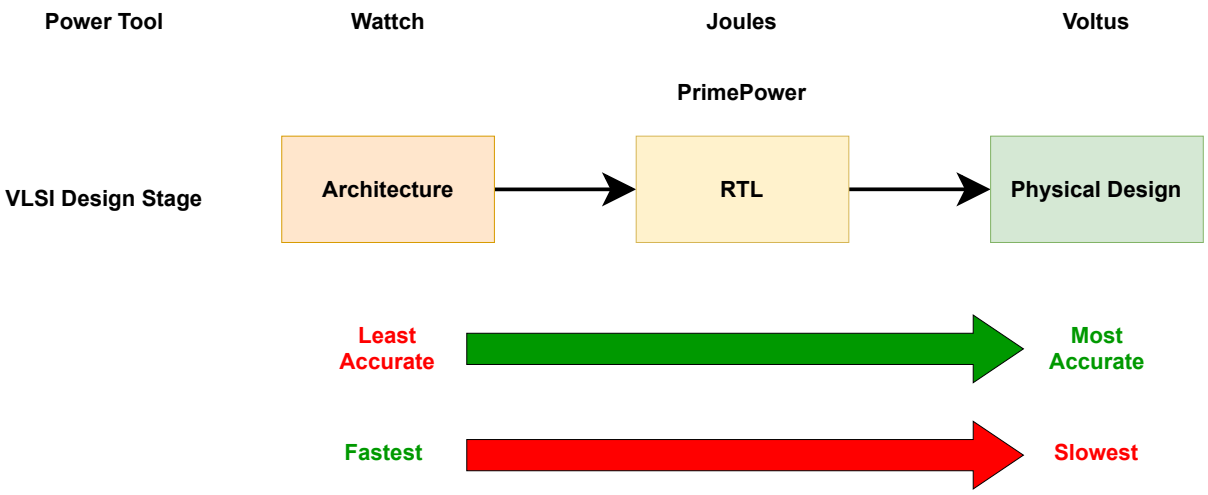


FIGURE 2.4: VLSI design stages and their available power estimation tools [2–5].

Unlike architectural energy models, these instruction-level energy models are created using dynamic energy values from real microarchitectural implementations, making them much more accurate to the energy consumption at signoff.

2.3.3 Power Estimation Tools

Power modeling for chip designs, in general, can span a wide range of levels in the VLSI design process, from architecture to GDSII file. Power models closer to the architecture stage tend to be faster, allowing for quick iteration over architectural design choices, at the cost of being one of the least signoff-accurate models. In contrast, power models closer to the GDSII file tend to be significantly slower at producing power estimates, but they are closer to actual signoff numbers.

The accuracy and latency of these power models are influenced by the input data to these models. Hardware designs closer to GDSII files can provide more traits and behaviors representative of the physical chip that higher-level power models cannot capture. However, as designs are converted into GDSII files, their size and complexity increase substantially. As a result, power models that take these designs as input can take significantly longer to process. Furthermore, in order to use these lower-level power models, additional time must be spent to convert hardware designs into an input accepted by the power model.

For this work, we use Joules [3], a state-of-the-art industry power estimation tool, to provide ground-truth dynamic energy values for software workloads running on DNN accelerators. Joules is an RTL-level power estimator that traces signals in RTL simulation on logic gate representations to determine capacitive loads and switching activity for every hardware module in a design.

Joules strikes a balance between architectural and post-place-and-route power estimation tools, where the design input comes early enough in the VLSI design process, yet still provides critical microarchitectural details for the tool to make accurate estimates.

Chapter 3

Energy Model Overview

To demonstrate the methodology required to create an instruction-level energy model for DNN accelerators, we construct one for the Gemmini accelerator. To do this, we determine the EPI for all Gemmini instructions to create the energy model. For validation, we count the Gemmini instructions of compiled target workloads to predict their overall energy consumption with the precomputed EPI values. As an analytical model, these dynamic energy predictions are delivered almost instantaneously, similar to many other architectural energy models.

We design two energy models, a naive ISA-based model and an optimized dimension-aware instruction-level (DAIL) model. The ISA-based energy model is inspired by energy models for CPUs and serves as the de facto baseline model for this work. The DAIL energy model is built on top of the ISA-based model, optimizing for DNN accelerator-specific architectural states. Although the ISA-based energy model works effectively for CPUs, DNN accelerators face different challenges. The DAIL model considers these challenges for its energy predictions, making it an instruction-level energy model specialized for DNN accelerators.

3.1 Hardware Configuration

The SoC designed to evaluate Gemmini's energy efficiency uses the Intel16 fabrication process with a 250 MHz clock frequency to capture the traits and behaviors of a modern process technology, especially with respect to power consumption. Using the Chipyard ecosystem, the SoC is configured to be as simple as possible, containing:

- 1× In-order scalar RV64GC Rocket CPU

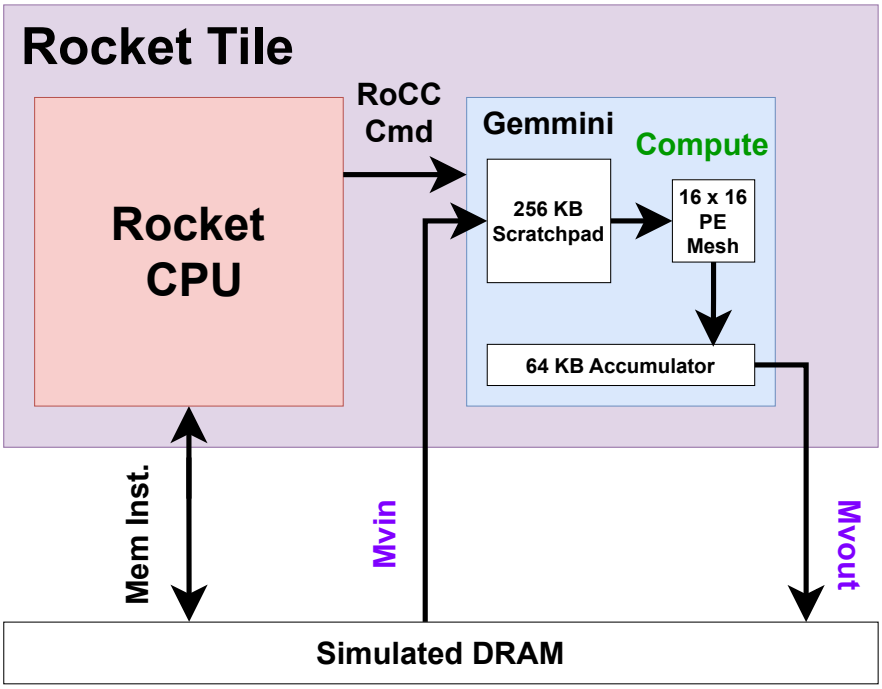


FIGURE 3.1: Gemmini operations on our SoC with the WS dataflow.

- 1× Gemmini accelerator that is coupled to the Rocket core
- Crossbar bus that uses the TileLink [26] protocol
- Off-chip DRAM simulated with DRAMSim2 [27]
- No last-level cache memory

Gemmini, specifically, is configured to have the following hardware properties:

- 16 × 16 systolic array
- 256 KB int8 scratchpad memory
- 64 KB int32 accumulator memory

In this SoC, Gemmini is interfaced through the host Rocket CPU that can issue Gemmini instructions, through rocket co-processor (RoCC) commands, in addition to standard RISC-V instructions. To manage data movement, the crossbar facilitates memory requests between Gemmini and simulated DRAM. Gemmini does not interface with Rocket's L1 DCache, and the SoC's last-level cache is removed from the memory hierarchy to prevent cache hits and misses from having an impact on the latency and, by extension, energy consumption of Gemmini data movement instructions.

3.2 Model Construction

TABLE 3.1: Matrix dimension impact on Gemmini's dynamic power for data movement.

Instruction	Matrix Dimensions	Bandwidth	Avg. Dynamic Power
mvin	16 rows, 16 columns	256 elems / instruction	1.048 mW
mvin	16 rows, 64 columns	1024 elems / instruction	5.064 mW

TABLE 3.2: Matrix dimension impact on Gemmini's dynamic power for computation.

Instruction	Matrix Dimensions	MAC Operations	Avg. Dynamic Power
compute_preloaded	A: 16 × 16, B: 16 × 16	4096 ops / instruction	23.892 mW
compute_preloaded	A: 2 × 16, B: 16 × 16	512 ops / instruction	13.142 mW
compute_preloaded	A: 16 × 16, B: 16 × 2	512 ops / instruction	15.288 mW
compute_preloaded	A: 16 × 2, B: 2 × 16	512 ops / instruction	16.243 mW
compute_preloaded	A: 2 × 16, B: 16 × 2	64 ops / instruction	8.617 mW

We present two energy models to characterize the EPI for all non-configuration Gemmini instructions with a single hardware configuration: an ISA-based energy model serving as a baseline model and a dimension-aware instruction-level (DAIL) energy model optimized for DNN accelerators. The ISA-based energy model uses the same methodology as CPU ISA-based energy models, but applies it to DNN accelerator ISAs. This model only considers EPI values based solely on instruction type (see Table 2.1 for a list of instructions). Since DNN accelerator ISAs are significantly smaller than CPU ISAs, the DAIL energy model considers multiple EPI values for the same instruction type, focusing on the energy impact of different matrix dimensions specified in the instruction arguments. Matrix dimension sizes can affect DNN accelerator energy consumption because they have a direct impact on MAC utilization and data movement bandwidth. Tables 3.1 and 3.2 show the importance of matrix dimensions to DNN accelerator dynamic power consumption.

To make these Gemmini energy models more generalizable, we only predict the dynamic energy of modules that are common across other DNN accelerator architectures, including the scratchpad and accumulator for data movement and mesh—the systolic array—for computation. As shown in Figure 3.2, these three modules make up 85.7% of Gemmini's average dynamic energy across various workloads. Gemmini's exclusive architectural features—including the internal DMA, ROB, and TLB modules—are not considered for this energy model.

3.2.1 Microbenchmarks

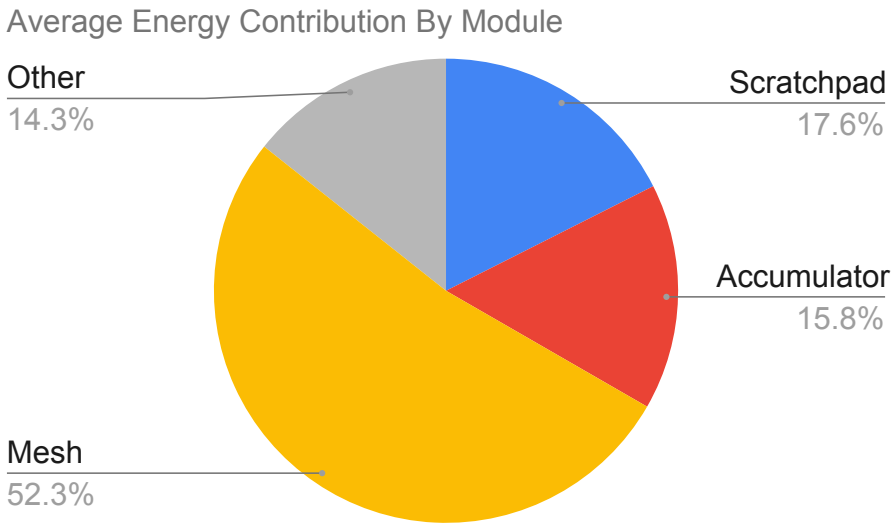


FIGURE 3.2: Average dynamic energy contribution per Gemmini module for target workloads.

```
int8_t A[DIM_I][DIM_K];
int8_t B[DIM_K][DIM_J];
int8_t C[DIM_I][DIM_J];

unsigned long cycle_start, cycle_end;
// Setup Code Here

cycle_start = read_cycles();
// Main Loop
for(int i = 0; i < 1000; i++) {
    // Gemmini Instructions Here
}
cycle_end = read_cycles();
```

LISTING 3.1: Microbenchmark code format

To characterize the dynamic energy of Gemmini's instructions, we construct microbenchmarks that repeatedly execute the same Gemmini instruction 1,000 times to produce a clear and consistent activity period in which to extract EPI per module. Once the dynamic energy of the microbenchmark is computed per module, the EPI can be calculated by dividing these energy values by 1,000. To simplify the microbenchmarks:

- Gemmini is set to the WS dataflow
- D (bias) is excluded from the GEMM operation

- No activation functions are used
- All data movement transactions use int8 matrix elements
  - For mvout instructions, which work with int32 accumulator data, values outside of the int8 range are saturated to be 127 or -128

As shown in Table 3.3, there are four types of instructions that we evaluate: mvin, mvout, preload and compute, and compute accumulated.

TABLE 3.3: Instructions used for microbenchmarks and their instruction arguments.

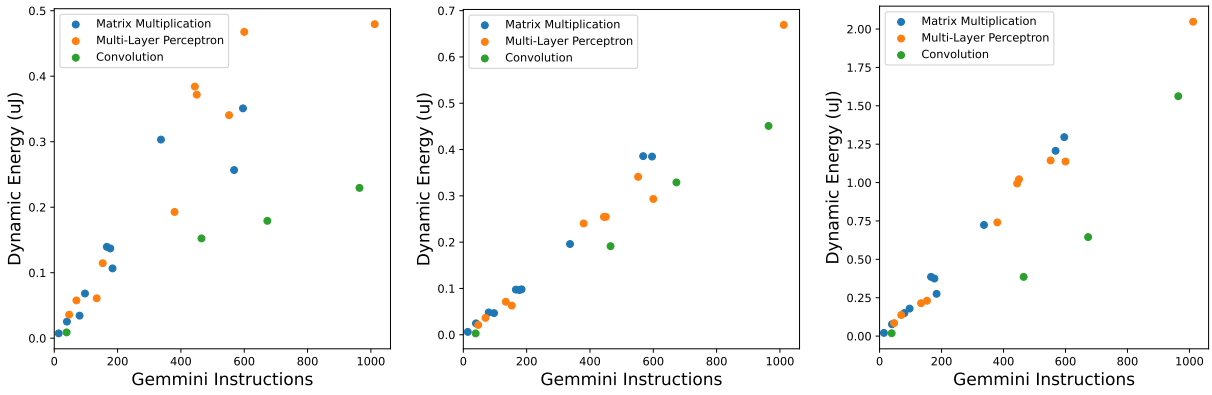
Instruction Type	Key Instruction Arguments
Mvin	mvin(src_addr, dest_sp_addr, cols, rows)
Mvout	mvout(src_acc_addr, dest_addr, cols, rows)
Preload and Compute	preload(B_sp_addr, C_acc_addr, B_cols, B_rows, C_cols, C_rows) compute_preloaded(A_sp_addr, A_cols, A_rows)
Compute Accumulated	preload(NULL, C_acc_addr, 0, 0, C_cols, C_rows) compute_accumulated(A_sp_addr, A_cols, A_rows)

3.2.1.1 Mvin Microbenchmark

The mvin microbenchmarks move randomized int8 data into the scratchpad using the mvin instruction. For setup, the scratchpad is completely filled with randomized int8 data. In the main loop, the mvin instruction is given a new scratchpad address on each iteration. Multiple microbenchmarks are created to evaluate the energy impact of moving in different numbers of columns, ranging from 1 column to 64 columns, while the number of rows is kept constant at 16.

3.2.1.2 Mvout Microbenchmark

The mvout microbenchmarks move randomized, saturated int8 data out of the accumulator and into main memory using the mvout instruction. For setup, the accumulator is completely filled with randomized int32 data. In the main loop, the mvout instruction is given a new accumulator address on each iteration. Multiple microbenchmarks are created to evaluate the energy impact of moving out different numbers of columns, ranging from 1 column to 64 columns, while the number of rows is kept constant at 16.



(A) Scratchpad (B) Accumulator (C) Mesh

FIGURE 3.3: Gemmini instruction count and ground-truth dynamic energy per target workload.

3.2.1.3 Preload and Compute Microbenchmark

The preload and compute microbenchmarks do non-tiled matrix multiplications on randomized A and B int8 matrices. For setup, the randomized A and B matrices are moved into the scratchpad. In the main loop, the preload instruction sets the same output accumulator address for C and reloads the same B matrix data into the mesh. Then, the compute_preloaded instruction reloads the same A matrix data into the mesh and recomputes C. Multiple microbenchmarks are created to evaluate the energy impact of moving out different numbers of A rows, A columns (B rows), and B columns, with each argument ranging from 1 to 16.

3.2.1.4 Compute Accumulated Microbenchmark

The compute accumulated microbenchmarks do non-tiled matrix multiplications on randomized A and B int8 matrices. For setup, the randomized A matrix is moved into the scratchpad while the B matrix is loaded into the mesh. In the main loop, the preload instruction sets the same output accumulator address for C, but does not load a new matrix into the mesh. Then, the compute_accumulated instruction reloads the same A matrix data into the mesh and recomputes C with the reused B matrix. Multiple microbenchmarks are created to evaluate the energy impact of moving out different numbers of A rows, A columns (B rows), and B columns, with each argument ranging from 1 to 16.

3.2.2 Target Workloads

We create validation benchmarks comprised of matrix multiplication and convolution workloads so that we can compare their ground-truth dynamic energy against the predictions of our energy

models. Similar to the microbenchmark configuration, these target workloads use the WS dataflow, exclude bias from computation, do not use activation functions, and only use INT8 matrix elements for data movement. Workloads are separated into three classes:

- Single Matrix Multiplication
- Sequential Matrix Multiplication (Multi-Layer Perception Model)
- Multi-Channel 2-D Convolution

To ensure a variety of workload sizes and types, we create multiple benchmarks—each with different matrix dimensions—for every workload class. For matrix multiplication workloads, the A and B matrix sizes are unique per workload. For multi-layer perceptron workloads, the hidden layer count, input layer dimensions, output layer dimensions, and hidden layer dimensions are unique per workload. For convolution workloads, the input channel count, output channel count, input matrix dimensions, and kernel matrix dimensions are unique for every workload.

3.2.3 ISA-based Energy Model

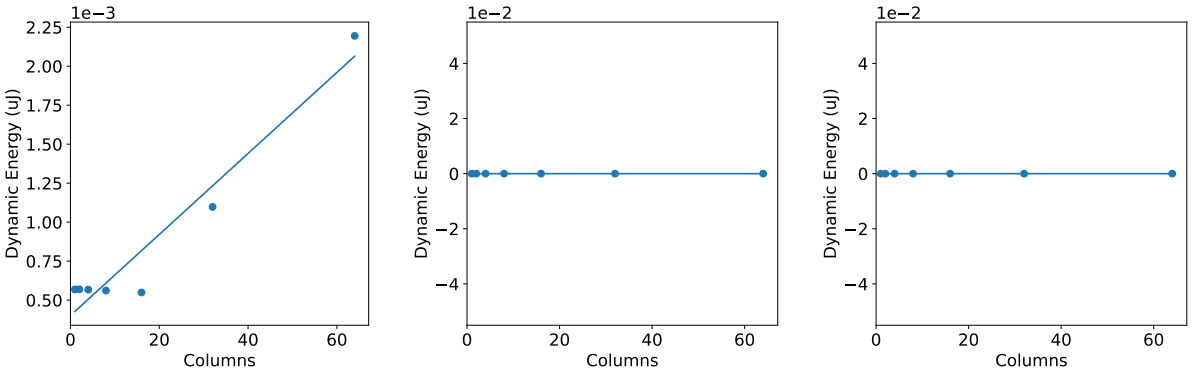
TABLE 3.4: Arguments used for each instruction type for ISA-based microbenchmarks.

Instruction Type	Instruction Arguments
Mvin	16 rows, 64 columns
Mvout	16 rows, 64 columns
Preload and Compute	A: 16×16, B: 16×16
Compute Accumulated	A: 16×16, B: 16×16

The ISA-based energy model predicts the dynamic energy of the following modules:

- Scratchpad
- Accumulator
- PE Mesh (Systolic Array)

For each instruction type, only one corresponding microbenchmark is created to compute its EPI. As shown in Table 3.4, the ISA-based energy model assumes maximum bandwidth per data movement instruction and maximum MAC utilization per computation instruction. When evaluating the



(A) Scratchpad (B) Accumulator (C) Mesh

FIGURE 3.4: Regression for mvin instructions used to predict EPI for various matrix column sizes.

dynamic energy of validation benchmarks, every Gemmini instruction in the workload is counted to determine the amount of each Gemmini instruction type. Then, each instruction type multiplies its EPI, as determined by the microbenchmarks, with its instruction count in the workload. The validation benchmark's total dynamic energy is calculated by summing up the overall dynamic energy contribution by each Gemmini instruction type on a per-module basis.

$$\begin{gathered} \mathrm{E} = \sum_{i \in I} \sum_{m \in M} \mathrm{n}_i \times \mathrm{EPI}_{i,m} \\ I = \{\texttt{mvin}, \texttt{mvout}, \texttt{compute\_preloaded}, \texttt{compute\_accumulated}\} \\ M = \{\text{Scratchpad}, \text{Accumulator}, \text{PE Mesh}\} \end{gathered} \tag{3.1}$$

3.2.4 Dimension-Aware Instruction-Level (DAIL) Energy Model

TABLE 3.5: Matrix dimensions, from instruction arguments, and their associated EPI, derived from microbenchmarks, are used to train a linear regression model to predict EPI for any matrix size.

Instruction Type	Regression Variables
Mvin	rows, cols
Mvout	rows, cols
Preload and Compute	A_rows, A_cols, B_cols
Compute Accumulated	A_rows, A_cols, B_cols

Similar to the ISA-based energy model, the DAIL energy model also predicts the dynamic energy of the scratchpad, accumulator, and PE mesh modules. Since there is a significant energy difference between Gemmini instructions of the same type but with different matrix dimensions in the

instruction arguments, we use a linear regression model to determine the EPI of an individual instruction. For each instruction type, multiple microbenchmarks are created, each microbenchmark differing only in their instruction arguments. The linear regression model is trained on the microbenchmark's matrix dimensions and the corresponding EPI so that we can provide unique EPI estimates for any possible combination of arguments for that instruction type. Table 3.5 shows the regression parameters for each instruction type. An example of regression analysis can be seen in Figure 3.4 for mvin instructions. For validation benchmarks, dynamic energy prediction is nearly identical to the ISA-based energy model, but extra steps are required to determine an instruction type's dynamic energy contribution:

1. For each instruction type, sum up the dynamic energy contribution by each unique combination of arguments of that same instruction type
   (a) Instruction counting is done at the argument-level granularity, meaning two instructions can only be counted together if they have identical instruction types and arguments
   (b) An instruction's EPI is predicted from its respective instruction type's regression model, with its respective instruction arguments being fed as inputs to the model
   (c) Similar to the ISA-based energy model, an EPI regression model is created per module
2. Once dynamic energy contribution is determined for all instruction types, sum them up
   (a) This step is identical to how the ISA-based energy model does it

$$
\begin{gathered}
\mathrm{E} = \sum_{i \in I} \sum_{\text{args}} \sum_{m \in M} \mathrm{n}_{i(\text{args})} \times \mathrm{EPI}_{i,m(\text{args})} \\
I = \{\texttt{mvin}, \texttt{mvout}, \texttt{compute\_preloaded}, \texttt{compute\_accumulated}\} \\
\text{mvin and mvout args} \in \{(x, y) \in \mathbb{N} \times \mathbb{N} \mid 1 \leq x \leq 16,\ 1 \leq y \leq 64\} \\
\text{compute\_preloaded and compute\_accumulated args} \in \{(x, y, z) \in \mathbb{N}^3 \mid 1 \leq x, y, z \leq 16\} \\
M = \{\text{Scratchpad}, \text{Accumulator}, \text{PE Mesh}\}
\end{gathered}
\tag{3.2}
$$

Chapter 4

Instruction-Level Energy Model for Gemmini

4.1 Toolflow

A variety of industry and Chipyard ecosystem tools are utilized to create an instruction-level energy model for Gemmini. Joules estimations are utilized as the ground-truth dynamic energy to compare

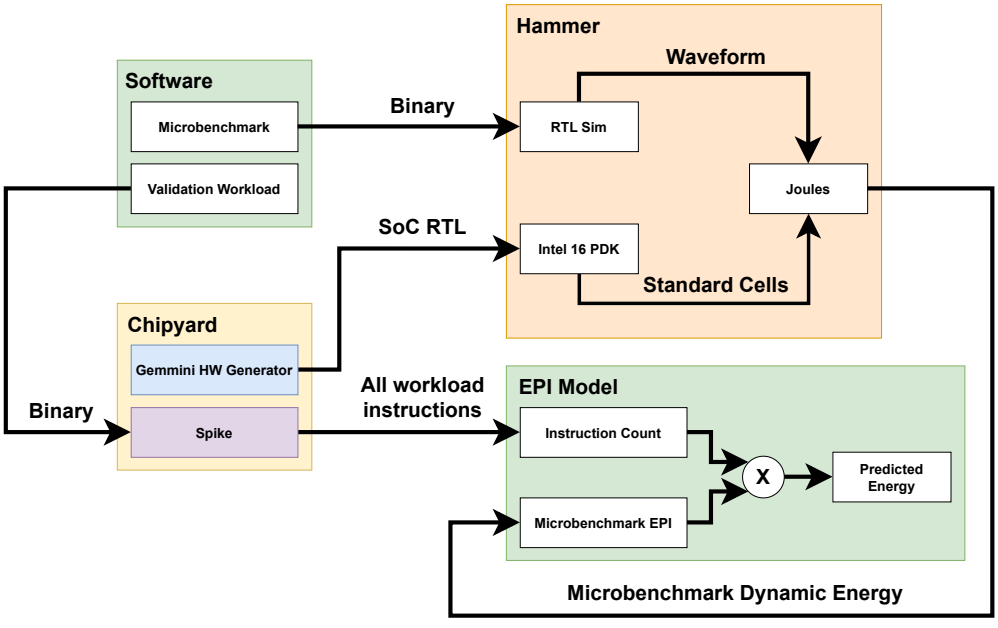


FIGURE 4.1: An architecture diagram of instruction-level energy modeling for Gemmini, utilizing Chipyard [6], Hammer [7], Spike [8], and Joules [3]

against our models. Spike [8], a RISC-V ISA simulator that supports Gemmini instructions, is utilized as a functional simulator to extract Gemmini instruction counts from target workloads. Hammer [7] is a Chipyard tool that generates TCL scripts to run RTL simulation and Joules power estimation. Figure 4.1 shows how all these tools interact with each other. The following are detailed steps for how instruction-level energy modeling works for Gemmini:

Model Creation

1. Chipyard generates RTL for the SoC configuration containing Gemmini
2. Microbenchmarks are compiled and run through RTL simulation to create a waveform
3. Microbenchmarks are run through Joules to determine dynamic energy consumption
    (a) Joules takes in RTL simulation waveforms and post-synthesis gate-level designs as inputs
4. EPI is computed for each module from microbenchmark dynamic energy consumption
5. (DAIL energy model only) Linear regression models are created to predict EPI for each instruction type based on instruction arguments for each module
    (a) This model is trained on EPI values from microbenchmarks of the same instruction type
    (b) Every module has its own EPI linear regression model

Model Validation

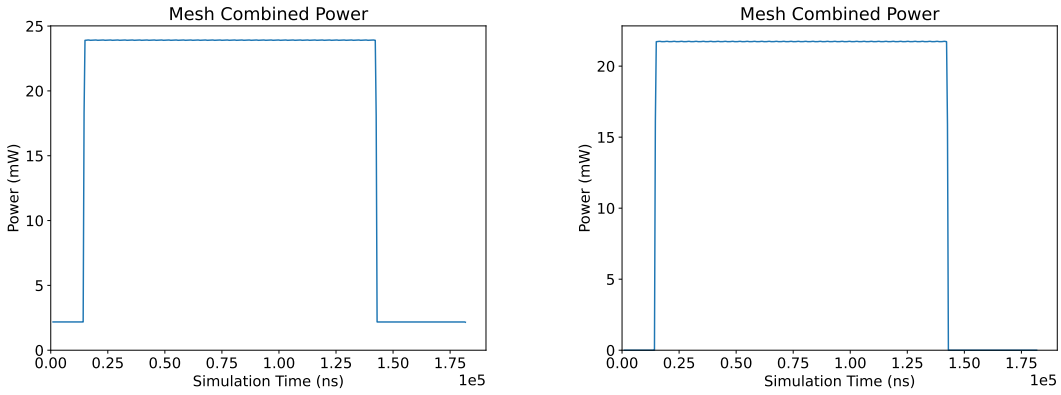
1. Validation benchmarks are compiled and run through Spike to generate a list of Gemmini instruction commits
2. For each instruction type and (DAIL energy model only) set of instruction arguments, multiply the instruction count with the pre-computed EPI from model creation for each module
    (a) The DAIL energy model uses the linear regression prediction as the EPI for an instruction

4.1.1 Dynamic Energy Computation

Joules provides detailed reports and plots that estimate the static and dynamic power consumption of software workloads over time. To determine the dynamic energy of microbenchmarks and validation benchmarks, we use Joules to create power plots for each module. Dynamic energy is computed for workloads by multiplying the average dynamic power across the entire power plot with its latency, also found in the power plot.

TABLE 4.1: Gemmini aggregated baseline power consumption by module.

Module	Baseline Power (mW)
Scratchpad	0.0494
Accumulator	1.5294
PE Mesh	2.1804
Gemmini	14.6065



(A) Original Power Plot (B) Baseline Power Removed

FIGURE 4.2: Power plot visualization for microbenchmarks.

Initial experimentation found that even software workloads without Gemmini instructions cause Gemmini to consume a non-negligible amount of both static and dynamic power. The presence of static power was expected, but these results showed that a small, non-negligible amount of activity occurs within Gemmini even without it being explicitly invoked via instructions. Thus, a baseline Gemmini power consumption that aggregates the static and baseline dynamic power consumption for each module was determined by running a simple program, without Gemmini instructions, through Joules. The exact power consumption values can be seen in Table 4.1.

With the baseline power for each Gemmini module determined, the dynamic power is computed for both microbenchmarks and validation benchmarks by subtracting the baseline power from their overall power consumption, eliminating both the static power consumption and baseline dynamic power consumption that persist regardless of the workload. Once a workload's power plot contains only dynamic power, its dynamic energy can be computed. Figure 4.2 shows the impact of removing baseline power from a microbenchmark's mesh module power plot.

$$P_{\text{baseline}} = P_{\text{static}} + P_{\text{dynamic\_baseline}} \text{ mW}$$

$$P_{\text{workload, dynamic}} = P_{\text{workload, joules}} - P_{\text{baseline}} \text{ mW}$$

TABLE 4.2: ISA-based energy model Mean Absolute Percentage Error (MAPE) and 95% confidence interval of validation workload dynamic energy predictions per module.

Validation Workload	Scratchpad	Accumulator	PE Mesh	Combined
Matrix Multiplication	0.31 ± 0.16	0.07 ± 0.04	0.10 ± 0.07	0.12 ± 0.07
Multi-Layer Perceptron	0.33 ± 0.12	0.07 ± 0.04	0.10 ± 0.07	0.11 ± 0.07
Convolution	2.05 ± 0.96	1.74 ± 2.58	1.36 ± 0.82	1.44 ± 1.09
Overall	0.61 ± 0.31	0.35 ± 0.46	0.31 ± 0.24	0.34 ± 0.26

TABLE 4.3: DAIL energy model Mean Absolute Percentage Error (MAPE) and 95% confidence interval of validation workload dynamic energy predictions per module.

Validation Workload	Scratchpad	Accumulator	PE Mesh	Combined
Matrix Multiplication	0.21 ± 0.11	0.04 ± 0.04	0.09 ± 0.06	0.07 ± 0.05
Multi-Layer Perceptron	0.18 ± 0.11	0.04 ± 0.03	0.11 ± 0.07	0.08 ± 0.06
Convolution	0.80 ± 0.36	0.10 ± 0.05	0.13 ± 0.11	0.24 ± 0.12
Overall	0.29 ± 0.12	0.05 ± 0.02	0.10 ± 0.05	0.10 ± 0.05

4.2 Model Evaluation

When validating against target workloads, the ISA-based energy model achieved a Mean Absolute Percentage Error (MAPE) of 34%, with a 95% confidence interval of [8%, 60%] for its dynamic energy predictions. On the other hand, the DAIL energy model achieved a MAPE of 10%, with a 95% confidence interval of [5%, 15%] for its dynamic energy predictions. MAPE per module is depicted in Table 4.2 for the ISA-based energy model and Table 4.3 for the DAIL energy model.

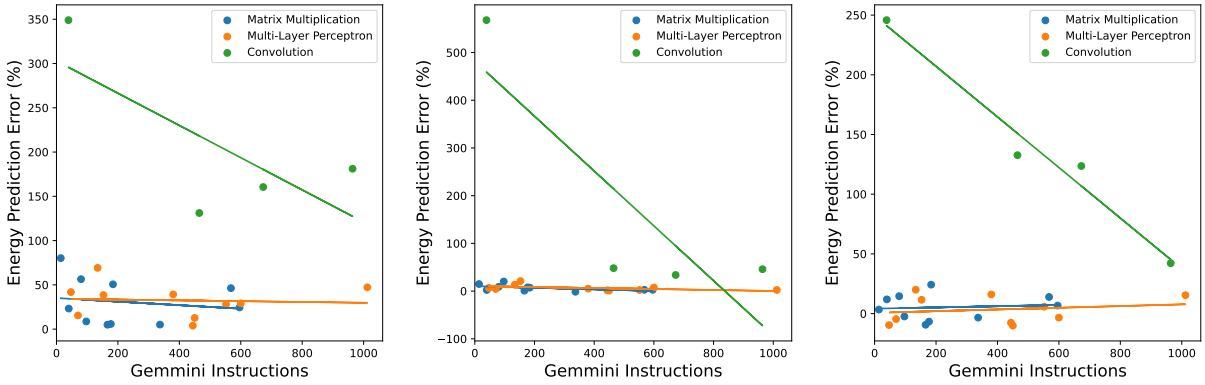
The effectiveness of the DAIL energy model over the baseline ISA-based energy model for Gemmini energy prediction shows that there is a significant distinction between the sub-fields of DNN accelerator energy modeling and CPU energy modeling. By specifically targeting DNN accelerator architectures, the DAIL energy model was able to better capture the behavior of Gemmini programs, especially lower utilization ones such as convolution workloads. Furthermore, the DAIL energy model slightly improved precision across all modules, effectively decreasing the range of the 95% confidence interval. These results show the effectiveness of creating instruction-level energy models for DNN accelerators through the methodology that we provide.

4.2.1 ISA-based Energy Model

Since the ISA-based energy model differentiated between Gemmini instructions only by type, one EPI value is assigned per instruction type. Instruction parameters used for these microbenchmarks are listed in Table 3.4 and EPI values per module are included in Table 4.4.

TABLE 4.4: ISA-based energy model's EPI for each instruction type per module.

Instruction Type	Spad EPI (uJ)	Acc EPI (uJ)	Mesh EPI (uJ)	Combined (uJ)
Mvin	2.19×10^{-3}	0	0	2.19×10^{-3}
Mvout	6.72×10^{-5}	4.98×10^{-4}	0	5.65×10^{-4}
Preload and Compute	5.59×10^{-4}	7.53×10^{-4}	2.73×10^{-3}	4.04×10^{-3}
Compute Accumulated	5.60×10^{-4}	7.60×10^{-4}	2.67×10^{-3}	3.99×10^{-3}



(A) Scratchpad (B) Accumulator (C) Mesh

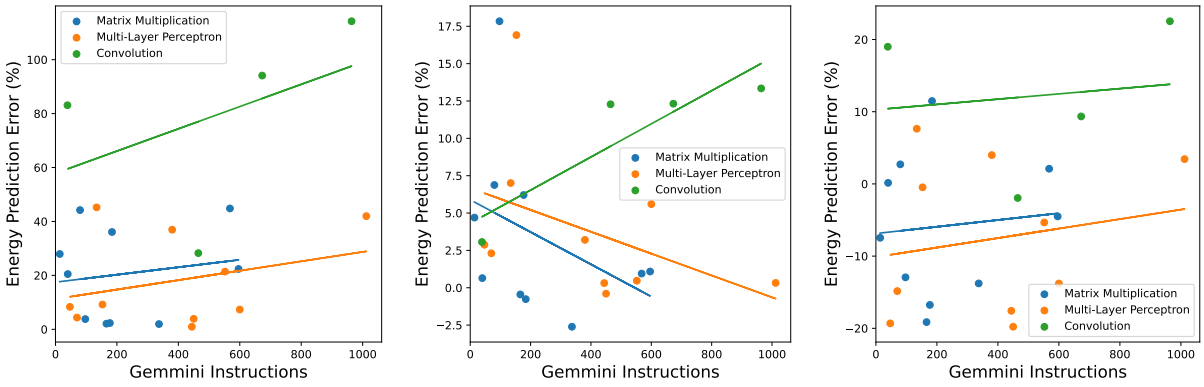
FIGURE 4.3: ISA-based energy model Mean Percentage Error (MPE) per module and Gemmini instruction counts for target workloads.

Experimental results show that generic ISA-based energy modeling, common for CPU designs, is much less effective for DNN accelerators, where MAC utilization and instruction bandwidth have a substantial impact on dynamic energy. With a high MAPE and spread, the ISA-based energy model faces considerable difficulties modeling the dynamic energy of DNN accelerators.

4.2.2 DAIL Energy Model

The DAIL energy model significantly improves dynamic energy predictions for convolution workloads, with a 6.0× reduction of MAPE and a 9.1× reduction of spread for the 95% confidence interval. Although the DAIL energy model improves prediction accuracy for the scratchpad module, it still possesses a high MAPE of 29% when compared to the MAPE of the accumulator (5%) and PE mesh (10%) modules. Despite this, the DAIL energy model improves upon the prediction accuracy of the baseline ISA-based energy model, with a 90% overall accuracy.

The major sources of error come from convolution workloads and the scratchpad module. These errors can compound, with scratchpad dynamic energy predictions for convolution workloads still possessing a MAPE of 80%, with a 95% confidence interval of [44%, 116%], when using the DAIL model. For convolution workloads, this error likely comes from its exceptionally low Gemmini utilization, meaning that the error would reduce with larger convolutions. For the scratchpad module,



(A) Scratchpad (B) Accumulator (C) Mesh

FIGURE 4.4: DAIL energy model Mean Percentage Error (MPE) per module and Gemmini instruction counts for target workloads.

TABLE 4.5: Times associated with data generation, training, and prediction.

	ISA-based Energy Model	DAIL Energy Model	Joules Evaluation
Data Generation Time	8 hours	90 hours	N/A
Training Time	76.71 ms	866.56 ms	N/A
Prediction Time	40.75 ms	41.41 ms	3 hours

this error likely comes from additional overhead from microbenchmarks that cause the scratchpad EPI to increase for some—if not all—Gemmini instructions, resulting in an over-prediction of overall scratchpad dynamic energy. Without changing the methodology, updating microbenchmarks that use the scratchpad would see a reduction in error.

4.3 Model Usability

The DAIL energy model for Gemmini has an accuracy close to Joules, the ground-truth dynamic energy evaluator, but it can predict dynamic energy consumption significantly faster. Table 4.5 shows the time for each of the three steps. Data generation time was evaluated on a single run for each due to the significant amount of time it took to generate all ground-truth dynamic energy data with Joules. Model training—building the instruction-level energy model with generated data—and prediction time—instruction counting and using the energy model—was averaged across 10 runs. Although there are significant differences in data generation and training time between the ISA-based and DAIL energy models, both models predict dynamic energy consumption significantly faster than Joules.

Chapter 5

Conclusion

In this work, we presented a methodology for designing instruction-level energy models for DNN accelerators. By creating instruction-level energy models for the Gemmini accelerator, including the ISA-based and DAIL energy models, we showed that this methodology can be applied to a real DNN accelerator. With the ISA-based energy model serving as a baseline energy model inspired by work from CPU energy modeling, we outperformed it with the DAIL energy model and reduced the overall benchmark MAPE by 3.4× and the 95% confidence interval range by 5.2×. For convolution workloads, where the ISA-based energy model significantly over-predicted dynamic energy consumption compared to other workloads, the DAIL energy model reduced the MAPE by 6.0× and the 95% confidence interval range by 9.1×.

The DAIL energy model, created to tackle the intricacies of DNN accelerator architectures, performs with an average accuracy of 90% with a 95% confidence interval of [85%, 95%]. The DAIL energy model's improvement over the baseline ISA-based energy model shows that CPU energy modeling techniques do not completely transfer over to DNN accelerator energy modeling. Furthermore, due to the speed and accuracy of the DAIL model, this work shows that DNN accelerator energy models can have the speed of analytical architectural models, like Timeloop, with an accuracy remarkably close to the accuracy of industry tools, like Joules. With this work, we bridge the gap between architectural and microarchitectural DNN accelerator energy models to empower software developers to make workloads for DNN accelerators more energy-efficient than ever before.

Bibliography

[1] Hasan Genc, Seah Kim, Alon Amid, Ameer Haj-Ali, Vighnesh Iyer, Pranav Prakash, Jerry Zhao, Daniel Grubb, Harrison Liew, Howard Mao, Albert Ou, Colin Schmidt, Samuel Steffl, John Wright, Ion Stoica, Jonathan Ragan-Kelley, Krste Asanovic, Borivoje Nikolic, and Yakun Sophia Shao. Gemmini: Enabling systematic deep-learning architecture evaluation via full-stack integration. In *2021 58th ACM/IEEE Design Automation Conference (DAC)*, pages 769–774, 2021. doi: 10.1109/DAC18074.2021.9586216.

[2] D. Brooks, V. Tiwari, and M. Martonosi. Wattch: a framework for architectural-level power analysis and optimizations. In *Proceedings of 27th International Symposium on Computer Architecture (IEEE Cat. No.RS00201)*, pages 83–94, 2000.

[3] Joules rtl power solution. URL `https://www.cadence.com/en_US/home/tools/digital-design-and-signoff/power-analysis/joules-rtl-power-solution.html`.

[4] Primepower: Rtl to signoff power analysis. URL `https://www.synopsys.com/implementation-and-signoff/signoff/primepower.html`.

[5] Voltus ic power integrity solution. URL `https://www.cadence.com/en_US/home/tools/digital-design-and-signoff/silicon-signoff/voltus-ic-power-integrity-solution.html`.

[6] Alon Amid, David Biancolin, Abraham Gonzalez, Daniel Grubb, Sagar Karandikar, Harrison Liew, Albert Magyar, Howard Mao, Albert Ou, Nathan Pemberton, Paul Rigge, Colin Schmidt, John Wright, Jerry Zhao, Yakun Sophia Shao, Krste Asanović, and Borivoje Nikolić. Chipyard: Integrated design, simulation, and implementation framework for custom socs. *IEEE Micro*, 40(4):10–21, jul 2020. ISSN 0272-1732. doi: 10.1109/MM.2020.2996616. URL `https://doi.org/10.1109/MM.2020.2996616`.

[7] Harrison Liew, Daniel Grubb, John Wright, Colin Schmidt, Nayiri Krzysztofowicz, Adam Izraelevitz, Edward Wang, Krste Asanović, Jonathan Bachrach, and Borivoje Nikolić. Hammer: a modular and reusable physical design flow tool: invited. In *Proceedings of the 59th*

ACM/IEEE Design Automation Conference, DAC '22, page 1335–1338, New York, NY, USA, 2022. Association for Computing Machinery. ISBN 9781450391429. doi: 10.1145/3489517.3530672. URL https://doi.org/10.1145/3489517.3530672.

[8] Màrius Montón. A risc-v systemc-tlm simulator, 2020. URL https://arxiv.org/abs/2010.10119.

[9] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet Classification with Deep Convolutional Neural Networks. In *Proceedings of the Conference on Neural Information Processing Systems (NeurIPS)*, 2012.

[10] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial networks, 2014.

[11] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep Residual Learning for Image Recognition. In *Proceedings of the Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[12] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Proceedings of the Conference on Neural Information Processing Systems (NeurIPS)*, 2017.

[13] OpenAI. Chatgpt. https://openai.com/blog/chat-gpt-3-launched/, 2020. Accessed: March 15, 2025.

[14] Yu-Hsin Chen, Tushar Krishna, Joel S. Emer, and Vivienne Sze. Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks. *IEEE Journal of Solid-State Circuits*, 52(1):127–138, 2017. doi: 10.1109/JSSC.2016.2616357.

[15] Ben Keller, Rangharajan Venkatesan, Steve Dai, Stephen G. Tell, Brian Zimmer, William J. Dally, C. Thomas Gray, and Brucek Khailany. A 17–95.6 tops/w deep learning inference accelerator with per-vector scaled 4-bit quantization for transformers in 5nm. In *2022 IEEE Symposium on VLSI Technology and Circuits (VLSI Technology and Circuits)*, pages 16–17, 2022. doi: 10.1109/VLSITechnologyandCir46769.2022.9830277.

[16] Pouya Houshmand, Giuseppe M. Sarda, Vikram Jain, Kodai Ueyoshi, Ioannis A. Papistas, Man Shi, Qilin Zheng, Debjyoti Bhattacharjee, Arindam Mallik, Peter Debacker, Diederik Verkest, and Marian Verhelst. Diana: An end-to-end hybrid digital and analog neural network soc for the edge. *IEEE Journal of Solid-State Circuits*, 58(1):203–215, 2023. doi: 10.1109/JSSC.2022.3214064.

[17] Gaofeng Zhou, Jianyang Zhou, and Haijun Lin. Research on nvidia deep learning accelerator. In *2018 12th IEEE International Conference on Anti-counterfeiting, Security, and Identification (ASID)*, pages 192–195, 2018. doi: 10.1109/ICASID.2018.8693202.

[18] Jonathan Bachrach, Huy Vo, Brian Richards, Yunsup Lee, Andrew Waterman, Rimas Avižienis, John Wawrzynek, and Krste Asanović. Chisel: Constructing hardware in a scala embedded language. In *DAC Design Automation Conference 2012*, pages 1212–1221, 2012. doi: 10.1145/2228360.2228584.

[19] Tien-Ju Yang, Yu-Hsin Chen, Joel Emer, and Vivienne Sze. A method to estimate the energy consumption of deep neural networks. In *2017 51st Asilomar Conference on Signals, Systems, and Computers*, pages 1916–1920, 2017. doi: 10.1109/ACSSC.2017.8335698.

[20] Angshuman Parashar, Priyanka Raina, Yakun Sophia Shao, Yu-Hsin Chen, Victor A. Ying, Anurag Mukkara, Rangharajan Venkatesan, Brucek Khailany, Stephen W. Keckler, and Joel Emer. Timeloop: A systematic approach to dnn accelerator evaluation. In *2019 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, pages 304–315, 2019. doi: 10.1109/ISPASS.2019.00042.

[21] Yannan Nellie Wu, Joel S. Emer, and Vivienne Sze. Accelergy: An architecture-level energy estimation methodology for accelerator designs. In *2019 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pages 1–8, 2019. doi: 10.1109/ICCAD45719.2019.8942149.

[22] Charles Hong, Qijing Huang, Grace Dinh, Mahesh Subedar, and Yakun Sophia Shao. Dosa: Differentiable model-based one-loop search for dnn accelerators. In *Proceedings of the 56th Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO '23, page 209–224, New York, NY, USA, 2023. Association for Computing Machinery. ISBN 9798400703294. doi: 10.1145/3613424.3623797. URL `https://doi.org/10.1145/3613424.3623797`.

[23] Yakun Sophia Shao and David Brooks. Energy characterization and instruction-level energy model of intel's xeon phi processor. In *International Symposium on Low Power Electronics and Design (ISLPED)*, pages 389–394, 2013. doi: 10.1109/ISLPED.2013.6629328.

[24] Krste Asanović, Rimas Avizienis, Jonathan Bachrach, Scott Beamer, David Biancolin, Christopher Celio, Henry Cook, Daniel Dabbelt, John Hauser, Adam Izraelevitz, Sagar Karandikar, Ben Keller, Donggyu Kim, John Koenig, Yunsup Lee, Eric Love, Martin Maas, Albert Magyar, Howard Mao, Miquel Moreto, Albert Ou, David A. Patterson, Brian Richards, Colin Schmidt, Stephen Twigg, Huy Vo, and Andrew Waterman. The rocket chip generator. Technical Report UCB/EECS-2016-17, Apr 2016. URL `http://www2.eecs.berkeley.edu/Pubs/TechRpts/2016/EECS-2016-17.html`.

[25] Zhiping Wang and W. Rhett Davis. An instruction-level power and energy model for the rocket chip generator. In *2021 IEEE/ACM International Symposium on Low Power Electronics and Design (ISLPED)*, pages 1–6, 2021. doi: 10.1109/ISLPED52811.2021.9502485.

[26] Size Zheng, Jin Fang, Xuegui Zheng, Qi Hou, Wenlei Bao, Ningxin Zheng, Ziheng Jiang, Dongyang Wang, Jianxi Ye, Haibin Lin, Li-Wen Chang, and Xin Liu. Tilelink: Generating efficient compute-communication overlapping kernels using tile-centric primitives, 2025. URL `https://arxiv.org/abs/2503.20313`.

[27] Paul Rosenfeld, Elliott Cooper-Balis, and Bruce Jacob. Dramsim2: A cycle accurate memory system simulator. *IEEE Computer Architecture Letters*, 10(1):16–19, 2011. doi: 10.1109/L-CA.2011.4.