Master of Science in Engineering: AI and Machine Learning
May 2026

Energy-Per-Inference Prediction for 1D CNNs on the ZCU104 FPGA

Marcus Fredriksson

Faculty of Computing, Blekinge Institute of Technology, 371 79 Karlskrona, Sweden

This thesis is submitted to the Faculty of Computing at Blekinge Institute of Technology in partial fulfillment of the requirements for the degree of Master of Science in Engineering: AI and Machine Learning. The thesis is equivalent to 20 weeks of full-time studies.

The author declares that he is the sole author of this thesis and has not used any sources other than those listed in the bibliography and identified as references. The author further declares that this thesis has not been submitted at any other institution to obtain a degree.

Contact Information:

Author:
Marcus Fredriksson
mafd21@student.bth.se
Department of Computer Science

University advisor:
Prof. Håkan Grahn
Department of Computer Science

Company advisor:
David Fredriksson, Delmius AB

Faculty of Computing
Blekinge Institute of Technology
SE–371 79 Karlskrona, Sweden

Internet : www.bth.se
Phone : +46 455 38 50 00
Fax : +46 455 38 50 57

Abstract

Background. Deploying a convolutional neural network on an FPGA for real-time industrial monitoring requires choosing among architectures that differ substantially in energy consumption, latency, and throughput. Determining the energy cost of a candidate architecture currently requires physical hardware access and extensive hardware benchmarking, making the exploration of large design spaces impractical.
Objectives. This thesis investigates whether energy-per-inference (EPI) can be predicted from model architecture parameters alone, before hardware benchmarking begins, enabling energy-aware model selection without repeated board programming.
Methods. A family of 23 deployable 1D CNN models spanning four orders of magnitude in parameter count was benchmarked on the AMD/Xilinx ZCU104 FPGA with a B4096 DPU configuration. A custom two-thread pipeline measured EPI, latency, and throughput across three independent repetitions. Gaussian process regression was trained after feature selection from a pool of 29 architecture-derived features, yielding one to two features per target.
Results. Two principal findings emerged. First, EPI follows a power-law scaling relationship ($E \propto P^{0.749}$, $R^2 = 0.984$), with sublinear latency scaling and a measured system-level power plateau at large model sizes. Together, these observations are consistent with a structural mismatch between image-optimized DPU hardware and short 1D time-series workloads. The GPR achieves lower-bound MdAPE values of 13.8%, 12.6%, and 15.4% for EPI, latency, and throughput respectively, due to partial nesting of feature selection. Empirical coverage of the stated 95% prediction intervals is 95.7%, consistent with the nominal level at this sample size. Second, a previously undocumented compiler deployability boundary was discovered in Vitis AI, where structurally identical models can fail to compile to a single DPU subgraph based solely on channel width, an effect not evident to the user before compilation is attempted.
Conclusions. EPI can be predicted from model architecture parameters alone, before hardware benchmarking begins, with screening-level accuracy for architecture ranking and shortlisting. Inspection of compiled binaries indicates an undocumented Vitis AI channel-width boundary. Above this boundary, standard Conv1d operations may be represented as operator decompositions that exceed the DPU instruction set boundary, causing CPU fallback that is not evident before compilation is attempted. Both findings are practically actionable. The predictor narrows the candidate architecture set before hardware access, and the compiler boundary makes empirical pre-flight verification a required step in any Vitis AI deployment pipeline. All measurement data and compiled model binaries are publicly released on Zenodo.

Keywords: FPGA, DPU, Energy-Per-Inference, Vitis AI, 1D CNN

Sammanfattning

Bakgrund. Att implementera ett konvolutionellt neuralt nätverk på en FPGA för industriell övervakning i realtid kräver val mellan arkitekturer som skiljer sig avsevärt i energiförbrukning, latens och genomströmning. Att fastställa energikostnaden kräver i dagsläget tillgång till fysisk hårdvara samt omfattande prestandamätningar, vilket gör det opraktiskt att utforska stora designrymder.
Mål. Denna studie undersöker om energiförbrukning per inferens (EPI) kan förutsägas enbart utifrån modellens arkitekturparametrar, innan hårdvarumätningar påbörjas, för att möjliggöra energimedvetet modellval utan upprepade hårdvarumätningar.
Metod. En uppsättning om 23 implementerbara 1D-CNN-modeller, som spänner över fyra storleksordningar i antal parametrar, utvärderades på en AMD/Xilinx ZCU104 FPGA med en B4096 DPU-konfiguration. Ett tvåtrådat program mätte EPI, latens och genomströmning över tre oberoende körningar. Gaussisk processregression tränades efter egenskapsselektion från en pool av 29 egenskaper, vilket resulterade i en till två egenskaper per målvariabel.
Resultat. Två huvudsakliga resultat framkom. För det första följer EPI en potenslag ($E \propto P^{0.749}$, $R^2 = 0{,}984$), med sublinjär latensskalning och en uppmätt effektplatå vid stora modellstorlekar. Detta är förenligt med en strukturell obalans mellan bildoptimerad DPU-hårdvara och korta 1D-tidsserier. GPR-modellen uppnår undre gränser för MdAPE på 13,8%, 12,6% respektive 15,4% för EPI, latens och genomströmning, till följd av partiellt nästad egenskapsselektion. Den empiriska täckningsgraden på 95,7% för de angivna 95%-prediktionsintervallen är förenlig med den nominella nivån vid $N = 23$. För det andra identifierades en tidigare odokumenterad gräns för kompilerbarhet i Vitis AI där strukturellt identiska modeller kan misslyckas att kompileras till en enda DPU-delgraf enbart beroende på kanalbredd.
Slutsatser. EPI kan förutsägas enbart utifrån modellens arkitekturparametrar, innan hårdvarumätningar påbörjas, med noggrannhet på en nivå som stödjer urval och rangordning av arkitekturer. Inspektion av kompilerade binärer indikerar en odokumenterad Vitis AI-gräns för kanalbredd, där Conv1d-operationer kan representeras av operatoruppdelningar som överskrider DPU:ns instruktionsgränser. Detta leder till CPU-fallback som inte är synligt innan kompilering har genomförts. Båda resultaten är praktiskt användbara. Prediktorn minskar antalet kandidatarkitekturer innan hårdvarutillgång krävs, och kompilatorbegränsningen innebär att empirisk förhandsverifiering bör vara ett obligatoriskt steg i varje Vitis AI-baserad distributionspipeline. Samtliga mätdata och kompilerade modellbinärer är offentligt tillgängliga via Zenodo.

Nyckelord: FPGA, DPU, Energi per inferens, Vitis AI, 1D CNN

Acknowledgments

I would like to thank my university supervisor Håkan Grahn for his guidance and support throughout this project. His genuine interest in the topic and his willingness to engage with the technical details made working on this thesis a rewarding experience, and his constructive feedback shaped the work in ways that would not have been possible on my own.

I would also like to thank my company advisor David Fredriksson at Delmius, who grounded this thesis in a real engineering problem, contributed to the mathematics and science behind the simulation, and financed the work. His practical perspective and domain knowledge have been invaluable to this project.

Contents

Terminology

Batch size The number of input samples processed together in one inference call.

Channel width The number of output channels in a convolutional layer. One of the main architectural scaling variables in this thesis.

CNN Convolutional Neural Network. A neural network architecture based on learned convolutional filters. This thesis uses 1D CNNs for time-series inference.

CPU fallback A deployment condition where part of a compiled neural network graph executes on the CPU instead of the DPU. CPU fallback invalidates direct energy and latency comparisons with DPU-only models.

Depth The number of convolutional layers in a neural network. One of the main architectural scaling variables in this thesis.

DPU Deep Learning Processing Unit. The AMD/Xilinx pre-synthesized neural network accelerator implemented in the FPGA programmable logic.

EPI Energy-per-inference. The energy consumed to perform one neural network inference, reported in joules or microjoules.

FPGA Field-Programmable Gate Array. A reconfigurable integrated circuit whose hardware logic can be programmed after manufacturing.

PS/PL Processing System / Programmable Logic. In the ZCU104, the PS refers to the ARM-based processor subsystem, while the PL refers to the FPGA fabric where the DPU is implemented. The term is used primarily when discussing power rails that cover both domains.

pout1 The IRPS5401 output power rail used for EPI measurements in this thesis. It covers the main PS/PL power domain rather than the DPU alone.

GPR Gaussian Process Regression. A probabilistic regression method that predicts both a mean value and an uncertainty estimate.

INT8 8-bit integer representation. The quantized numerical format targeted by the DPU in this thesis.

Latency The time required to complete one inference.

LOO-CV Leave-One-Out Cross-Validation. A validation procedure where each data point is tested once while the model is trained on the remaining data.

MAC Multiply-Accumulate operation. A multiplication followed by an addition, used to estimate neural network compute cost.

MAPE Mean Absolute Percentage Error. The mean prediction error expressed as a percentage of the true value.

MdAPE Median Absolute Percentage Error. The median percentage prediction error, used to describe typical error with less sensitivity to outliers.

ONNX Open Neural Network Exchange. A model exchange format used to represent trained neural networks across different tools and runtimes.

Quantization The conversion of a neural network from high-precision arithmetic to lower-precision arithmetic for efficient hardware execution.

Subgraph A partition of a compiled neural network graph assigned to a specific execution target, such as the DPU or CPU.

sysfs A Linux virtual filesystem that exposes hardware and kernel state as readable file paths.

Throughput The number of inferences completed per unit time, typically reported as inferences per second.

VART Vitis AI Runtime. The runtime library used to execute compiled `.xmodel` files on the DPU.

Vitis AI The AMD/Xilinx toolchain used to quantize, compile, and deploy neural network models to DPU platforms.

XRT Xilinx Runtime. The runtime stack that manages communication between the processing system and FPGA programmable logic.

ZCU104 The AMD/Xilinx FPGA evaluation board used as the hardware platform in this thesis.

Chapter 1

Introduction

1.1 Background

Embedded artificial intelligence makes it practical to run neural network inference directly on edge hardware, eliminating the need to transmit raw sensor data to a remote server. Consider a rubber bushing test rig in an automotive development facility where the bushing is subjected to cyclic mechanical loading while sensors record displacement, velocity, and force at 1000 Hz. All product development relies on high-quality test data, so anomalies caused by faults, unexpected boundary conditions, or sensor issues must be detected in real time so that corrective action can be taken immediately rather than discovered in post-processing. Field-programmable gate arrays (FPGAs) are a compelling platform for this kind of inference task because they combine the programmability of a general-purpose processor with energy efficiency approaching that of custom silicon [15].

Despite mature toolchain support, deploying a CNN on an FPGA involves a non-trivial compilation and quantization pipeline that transforms a trained model into hardware-executable instructions. The critical gap is that determining how much energy each inference will consume on the target hardware currently requires physical board access and extensive benchmarking. Energy-per-inference (EPI), measured in joules (J) or microjoules (μJ), directly determines battery life and thermal budget at a given inference rate, yet no established method exists for predicting it from model architecture parameters before hardware access is available.

This thesis addresses that gap. A controlled benchmarking study across a family of 1D CNN models on the AMD/Xilinx ZCU104 FPGA establishes empirical EPI, latency, and throughput measurements, and a predictive model maps architecture parameters directly to these quantities. The study also uncovered a compiler-level deployability risk specific to the Vitis AI toolchain, where structurally identical models can produce incompatible compiled binaries depending solely on channel width.

1.2 Scope

This project is scoped to the deployment and energy characterization of CNN models on a single hardware platform, the AMD/Xilinx ZCU104, using the B4096 DPU configuration. No other FPGA platforms or DPU variants are considered, and no claims of generalization beyond this configuration are made.

The benchmark family consists of 1D CNN architectures trained in PyTorch [19]. Models up to `model-88m` are trained from scratch, while `model-132m` and `model-200m`

use randomly initialized weights and are included for deployment-level energy, latency, throughput, and compiler-limit characterization only. All models must be executable as a single DPU subgraph. Model accuracy on the benchmarking task is not an optimization target and the models are designed to cover a diverse range of architectural configurations rather than to maximize predictive performance on the underlying dataset. Quantization is fixed to INT8 throughout, as this is the quantization scheme produced by the Vitis AI toolchain, and no alternative quantization levels are explored.

The framework is limited to prediction and does not perform optimization. Given a set of model and hardware configuration parameters, it produces estimates of energy-per-inference, throughput, and latency. The primary goal of the predictive model is to demonstrate the feasibility of deployment-level EPI prediction from configuration parameters, rather than to achieve a specific prediction accuracy target or to identify a single best-performing modeling approach. The suitability of different predictive modeling strategies is explored empirically, guided by the characteristics of the collected dataset.

1.3 Aim

The aim of this project is to demonstrate the feasibility of predicting energy-per-inference for CNN models deployed on FPGA hardware using the ZCU104 platform as a reference. This is done through a data-driven approach grounded in empirical hardware measurements, resulting in a predictive model and publicly released datasets deposited on Zenodo [8–10] that allow the research community to build on this work. By connecting model-level design choices to observed energy, throughput, and latency outcomes, the project seeks to establish whether deployment-level EPI prediction is a viable basis for identifying well-suited model configurations for a given hardware platform without repeated hardware benchmarking.

1.4 Objectives

To achieve the aim, the project is structured around the following objectives.

- **Benchmarking and Data Collection**. Develop a replicable and transparent benchmarking procedure to measure energy-per-inference, latency, and throughput for CNN models deployed on the ZCU104 FPGA using the Vitis AI toolchain and B4096 DPU configuration, and collect structured performance data across a diverse range of model configurations.
- **Analysis**. Identify and characterize measurable relationships between CNN architecture parameters and observed energy-per-inference, throughput, and latency on the target hardware platform.
- **Modeling**. Construct a predictive model capable of estimating energy-per-inference, throughput, and latency from model configuration parameters, and assess its utility for deployment-level decision-making without repeated hardware synthesis.

- **Evaluation**. Evaluate the prediction accuracy of the model against empirical measurements and validate the benchmarking pipeline against a real-time workload to verify correctness and repeatability.

1.5 Research Questions

Code	Question
RQ1	How can energy-per-inference be estimated for CNN models deployed on FPGA hardware, alongside throughput and latency, from model configuration parameters?
RQ2	What relationships exist between CNN architecture parameters and observed energy-per-inference on the ZCU104 platform?
RQ3	How accurately can deployment-level energy-per-inference be predicted from empirical data collected on a fixed FPGA hardware configuration?

1.6 Outline

Chapter 2 – Background Provides the technical context for the study, covering FPGA-based inference, the Vitis AI DPU toolchain, 1D CNNs for time-series, INT8 quantization, and energy-per-inference as a metric.

Chapter 3 – Related Work Reviews the relevant literature on FPGA-based CNN inference and energy modeling, and identifies the research gaps this thesis addresses.

Chapter 4 – Method Describes the experimental pipeline, including the simulation dataset, the SimNet model family, the benchmarking procedure, and the predictive modeling approach.

Chapter 5 – Results and Analysis Presents the empirical results, including the EPI scaling law, the compiler deployability finding, and the GPR prediction accuracy.

Chapter 6 – Discussion Interprets the findings in the context of the research questions and discusses the implications for DPU-based embedded AI deployment.

Chapter 7 – Conclusions and Future Work States the conclusions, answers the research questions directly, and outlines directions for future work.

Appendix Provides supplemental material including the full SimNet model family table, architecture feature definitions, and simulation parameter ranges.

Chapter 2

Background

This chapter provides the technical context needed to understand the experimental pipeline and results presented in subsequent chapters. It first introduces machine learning, neural network inference, and convolutional neural networks, before describing field-programmable gate arrays, the Vitis AI toolchain and DPU architecture, INT8 quantization, and the energy-per-inference metric.

2.1 Machine Learning and Neural Network Inference

Machine learning (ML) refers to methods that learn patterns from data rather than being programmed through explicit rules. In supervised learning, a model is trained on input-output examples and learns a mapping from input data to a target label or value [5, 11]. In this thesis, the input is a window of time-series sensor data, and the target is a binary anomaly label indicating whether an anomalous event is present in that window.

A neural network is a machine learning model composed of layers of parameterized functions. Each layer transforms its input into a new representation, and the parameters of these transformations are adjusted during training to reduce a loss function that measures the difference between predicted and true labels. For a classification task, the final layer produces a score or probability for each class. Training is usually performed using gradient-based optimization, where errors are propagated backward through the network and used to update the model parameters [11, 24].

It is important to distinguish between training and inference. Training is the offline process of adjusting the model parameters from data. Inference is the online process of applying an already trained model to new input data. This thesis concerns inference rather than training on the FPGA. The models are trained on a host workstation, converted into a deployable representation, and then executed on the ZCU104 FPGA to measure energy-per-inference, latency, and throughput. In this setting, the model parameters are fixed, and the main deployment question is how expensive each forward pass is on the target hardware.

2.2 Convolutional Neural Networks

Convolutional neural networks (CNNs) are neural networks designed to exploit local structure in data. Instead of connecting every input value to every output value, a convolutional layer applies a small learned filter across neighboring input positions.

The same filter weights are reused at many positions, a property known as weight sharing. This makes CNNs parameter-efficient and allows them to detect local patterns regardless of where they occur in the input [16, 17].

A convolutional layer produces one or more output channels, also called feature maps. Each output channel corresponds to a learned filter. Early layers typically detect simple local patterns, while deeper layers combine these into more abstract representations. CNNs often also include nonlinear activation functions, such as ReLU, which allow the network to represent nonlinear relationships, and pooling operations, which reduce the spatial or temporal resolution of the feature maps. Pooling reduces the amount of computation in later layers and increases the effective receptive field of the network.

The computational cost of CNN inference is dominated by multiply-accumulate (MAC) operations. Each convolution requires repeated multiplications between input values and learned filter weights, followed by additions. As a result, architecture parameters such as depth, channel width, kernel size, and sequence length strongly influence the number of operations that must be executed. These parameters are therefore directly relevant to latency, throughput, and energy consumption on hardware accelerators.

2.3 Field-Programmable Gate Arrays

Field-programmable gate arrays are integrated circuits whose logic is implemented in reconfigurable hardware rather than fixed silicon. This reconfigurability makes FPGAs a middle ground between general-purpose processors and application-specific integrated circuits (ASICs) as they offer substantially better energy efficiency than CPUs or GPUs for inference workloads, while remaining reprogrammable after manufacture. For real-time industrial sensing applications, FPGAs provide deterministic, low-latency computation at a fixed power budget, properties that general-purpose processors cannot reliably guarantee [15].

Deploying a CNN on an FPGA, however, is not a straightforward process. The model must be transformed from the 32-bit floating-point representation used during training into a form the hardware can execute efficiently. This requires dedicated compilation toolchains, hardware-specific quantization, and empirical validation that the resulting hardware implementation behaves correctly.

2.4 The Vitis AI Toolchain and DPU

The AMD/Xilinx Vitis AI toolchain [3] automates the deployment of quantized neural networks onto FPGA hardware. Its central component is the Deep Learning Processing Unit (DPU), a configurable accelerator IP core that runs inside the FPGA's programmable logic fabric and is specialized for the multiply-accumulate operations that dominate CNN inference. The DPU is parameterized by its peak throughput. The B4096 configuration used in this thesis is rated at up to 4,096 GOPS of INT8 throughput [1].

The toolchain operates in two stages. First, the `vai_q_pytorch` quantizer converts trained PyTorch models from 32-bit floating-point to 8-bit integer (INT8) arithmetic,

calibrating scale factors using a small set of representative inputs. Second, the `vai_c_xir` compiler maps the quantized graph to DPU instruction sequences for the target hardware configuration. The result is a binary `.xmodel` file that can be executed on the board via the Vitis AI Runtime (VART). The compilation step is not transparent, as the compiler applies internal algebraic transformations that may produce different operator decompositions depending on model structure and channel widths, with consequences for deployability that are not predictable from the model architecture alone.

2.5 INT8 Quantization

Training neural networks in 32-bit floating-point precision provides numerical stability and optimization convenience, but 32-bit arithmetic is energy-intensive and requires substantial memory bandwidth. Quantization [12] maps trained weights and activations from floating-point to lower-precision integers, typically INT8. The DPU hardware is optimized specifically for INT8 multiply-accumulate operations, which consume substantially less energy per operation than their floating-point equivalents.

The Vitis AI toolchain uses post-training quantization as the model is first trained to convergence in floating-point, then quantized using a calibration step that determines appropriate scale factors for each layer's activations. This calibration step can use either representative input data or dummy inputs. As discussed in Section 6.10, the choice of calibration inputs is a source of systematic uncertainty in absolute EPI values, though relative cross-model comparisons are expected to be unaffected.

2.6 Energy-Per-Inference as a Deployment Metric

Energy-per-inference (EPI), measured in joules (J) or microjoules (μJ), is the total energy consumed by the hardware system during a single forward pass of the neural network. It is the product of the system's power draw and the inference latency.

$$\text{EPI} = P \times T, \tag{2.1}$$

where P is the mean power draw during the inference window and T is the inference latency. EPI directly determines battery life and thermal budget at a given inference rate as a system running inference at f_{inf} inferences per second consumes $\text{EPI} \times f_{\text{inf}}$ watts in steady state.

Despite its operational relevance, EPI is rarely reported as a primary metric in the FPGA inference literature. Works that do report it typically measure it post-deployment as an observation, not as a design criterion that could be used to guide model selection. The central question of this thesis is whether EPI can instead be predicted from model architecture parameters before any hardware access is required.

Chapter 3

Related Work

Predicting the energy cost of a neural network deployment before touching the hardware requires two things that the existing literature has pursued separately. The first is empirical energy characterization on real FPGA platforms, and the second is machine learning models for FPGA power estimation. Several works have done the first well, reporting energy figures after deployment on boards similar or identical to the ZCU104 used here. Others have built accurate ML-based power predictors, but grounded in hardware design artifacts such as HLS reports and switching activity simulations that a practitioner running Vitis AI simply does not have access to. What has not been done is bring these two threads together at the deployment level, where the only available inputs are the model architecture and the compiled binary.

Khandelwal et al. [14] deploy quantized multi-layer perceptrons (MLPs) as an intrusion detection system for automotive controller area networks on the ZCU104 FPGA, the same board used in this thesis. They use the FINN framework [6] to generate custom hardware IP from a highly quantized network, achieving 0.12 ms latency and 0.25 mJ per inference. The paper is the only prior work that reports EPI on the ZCU104 and is therefore a direct calibration point for the energy scale of the present study. However, FINN generates a bespoke circuit for each model, requiring a complete implementation cycle before any energy figure is known. EPI is therefore an outcome of the design process, not an input to it. The approach also targets MLPs for network packet classification rather than 1D CNNs for sensor time-series, so the workload characteristics differ substantially from this thesis.

Özdil and Örs [30] deploy ResNet-18 for image classification on the Kria KV260 using Vitis AI, the same toolchain as this thesis, and systematically vary DPU configuration (B512 through B4096) and clock frequency. They find that resource utilization is constant across frequencies but that lower frequencies increase energy consumption, and that higher-MAC DPU configurations at higher frequencies give the best energy efficiency. This is the most methodologically similar prior work, as it uses Vitis AI on a DPU platform and treats energy efficiency as a primary outcome. The key difference is the axis of variation. Özdil and Örs vary hardware configuration while holding the model fixed, whereas this thesis varies model architecture while holding the hardware fixed. Neither study predicts energy for an unseen configuration, and both require re-running the hardware deployment to obtain a new energy figure.

Reddy et al. [23] implement a custom patch-wise MobileNet-V1 accelerator on the ZCU104 for synthetic aperture radar image classification, reporting throughput and energy efficiency in GOP/s/W. Like Khandelwal et al., the energy figures are post-deployment measurements with no methodology for estimating them in advance.

Unlike the other deployment studies, Reddy et al. build a hand-crafted custom accelerator rather than using the DPU, which gives tighter performance for that specific model but prevents the results from generalizing across architectures. This work is relevant as further evidence that the ZCU104 is a viable platform for energy-efficient embedded inference, and as a contrast to the standard Vitis AI DPU flow used in this thesis.

Kedia et al. [13] study runtime and energy across multiple DNN topologies and DPU configurations on a Xilinx FPGA, formulating a design space exploration strategy for systems running multiple DNN accelerators concurrently. It is one of the few works to treat energy as a first-class objective alongside accuracy and throughput in a DPU deployment context. The study is complementary to this thesis rather than directly competing with it. Kedia et al. ask which DPU configuration to use for a given model, while this thesis asks which model architecture to choose for a given DPU configuration. Neither asks how to predict energy before measurement.

Lin et al. [18] introduce HL-Pow, a machine learning framework that predicts FPGA power from features extracted at the high-level synthesis (HLS) stage, bypassing time-consuming register-transfer level (RTL) simulation. HL-Pow achieves 4.67% average prediction error relative to on-board measurement and includes a design space exploration algorithm that trades off latency against power. This is the closest prior work to this thesis in using ML for FPGA power prediction. The critical difference is the level of abstraction. HL-Pow requires an HLS description of the target circuit, but a practitioner deploying a trained neural network through the standard Vitis AI workflow does not expose the HLS representation to the user. HL-Pow's features are therefore inaccessible without modifying the toolchain, meaning the approach cannot be used in a standard deployment workflow. This thesis targets exactly that deployment level, using only architecture parameters available before the first board is programmed.

Wei et al. [25] address the generalization limit of HLS-based power models. Since features are FPGA-specific, a model trained on one device does not transfer to another. They propose a transfer learning approach that adapts an HLS-derived power model to a new target FPGA using a small number of additional measurements, demonstrating that cross-FPGA power estimation is feasible. This direction is directly relevant to the future work of this thesis. If the EPI power law preserves its structure across DPU configurations with only the exponent and intercept shifting, a similar transfer learning approach could adapt the GPR predictor to a new DPU variant with few calibration measurements. The abstraction level still differs, as Wei et al. use HLS features rather than deployment-level architecture parameters, but the transfer learning mechanism itself is a concrete technique worth adopting.

Chen et al. [7] extend HLS-based power prediction to unseen FPGAs by encoding design switching activities and FPGA architectural features into a Graph Neural Network. Their model, ATAPP, achieves 13.09% average error on unseen FPGAs, compared to over 40% for the best prior method, and is the most accurate and most generalizable FPGA power predictor in the literature to date. Like HL-Pow and Wei et al., ATAPP requires RTL-level information not exposed through Vitis AI. The 13.09% error on unseen FPGAs is nonetheless a useful reference point. The GPR in this thesis achieves 13.8% MdAPE on leave-one-out cross-validation within a fixed hardware configuration, suggesting that deployment-level prediction can approach

the accuracy of design-level methods when the hardware is held constant.

Table 3.1 compares the seven reviewed works across five dimensions, namely whether EPI is the primary energy metric, whether a predictive model is built, whether the work addresses 1D CNNs, whether it operates at deployment level, and whether it uses the ZCU104 or an equivalent Vitis AI DPU.

Table 3.1: Comparison of related works. EPI = energy-per-inference as primary metric. Pred. = predictive model for energy. 1D = uses 1D CNN. Dep. = deployment-level rather than design-level (HLS/RTL). Same HW = ZCU104 or equivalent Vitis AI DPU.

Work	**EPI**	**Pred.**	**1D**	**Dep.**	**Same HW**
Khandelwal et al. [14]	✓	×	×	✓	✓
Özdil and Örs [30]	×	×	×	✓	✓
Reddy et al. [23]	×	×	×	✓	✓
Kedia et al. [13]	×	×	×	✓	×
Lin et al. [18]	×	✓	×	×	×
Wei et al. [25]	×	✓	×	×	×
Chen et al. [7]	×	✓	×	×	×
This thesis	✓	✓	✓	✓	✓

No single prior work checks more than three of the five columns, and the combination of a predictive model with deployment-level operation has not been achieved by any of the reviewed works. The four deployment studies report energy only after hardware access. The three prediction studies operate at HLS level and are inaccessible in a standard Vitis AI workflow. The gap this thesis fills is the empty cell at the intersection of these two groups. Among the reviewed works, none estimates EPI from architecture parameters before hardware benchmarking begins, using only information available in a standard deployment pipeline. Beyond that methodological gap, none of the reviewed works study 1D CNNs for time-series data. All deployment studies use 2D CNNs for image classification, and all prediction studies target general HLS designs or image models. The energy behavior of 1D CNN families on DPU platforms is therefore documented here for the first time.

Chapter 4

Method

This chapter describes the experimental pipeline used to answer the three research questions. The pipeline consists of seven stages: (1) environment setup, establishing a repeatable hardware and software stack; (2) simulation dataset generation, producing labeled time-series data from a physics-based rubber bushing model; (3) model family construction and training, building a suite of one-dimensional convolutional neural network (1D CNN) models spanning four orders of magnitude in parameter count; (4) compilation, converting each model to a fixed-point binary for the target Deep Learning Processing Unit (DPU); (5) on-device benchmarking, running each compiled model on the ZCU104 field-programmable gate array (FPGA) and recording hardware sensor data; (6) data processing, cleaning the raw measurements and calculating energy-per-inference (EPI) from empirical sensor data; and (7) predictive modeling, fitting regressors that map architecture parameters to deployment-level energy, latency, and throughput.

Figure 4.1 provides an overview of the pipeline and the data artifacts produced at each stage.

4.1 Environment Setup

The project uses two machines. The host workstation runs Ubuntu 24.04.4 LTS on an AMD Ryzen 9 5950X (16-core) with 64 GB of DDR4 memory at 3600 MHz and an NVIDIA RTX 3090 (24 GB GDDR6X VRAM), used for model development, training, and compilation. The target board is the AMD/Xilinx ZCU104 evaluation platform [4] with the official PYNQ 3.0.1 image [2] (XRT 2.8.743, VART 2.5.0), providing the Xilinx Runtime (XRT) and Vitis AI Runtime (VART) inference stack, which hosts a B4096 DPU configuration [1] in its programmable logic fabric. The B4096 DPU is loaded at boot via a bitstream overlay.

Model development, training, and Open Neural Network Exchange (ONNX) export are performed in a separate Python 3.13 environment on the host using PyTorch 2.10.0 and ONNX opset 17. INT8 (8-bit integer) quantization and DPU compilation are performed inside the official Vitis AI Docker container [3] (image tag `xilinx/vitis-ai-pytorch-cpu:3.5.0`, Ubuntu 20.04, Python 3.8, PyTorch 1.13.1, `vai_q_pytorch` 3.5.0), which provides the `torch_quantizer` and `vai_c_xir` compiler. The training and compilation environments are intentionally separate. The training environment uses a recent PyTorch release for model development, while the Vitis AI container provides the fixed toolchain required for quantization and compilation. The ZCU104 is connected to the host over a local network and accessed via SSH for file

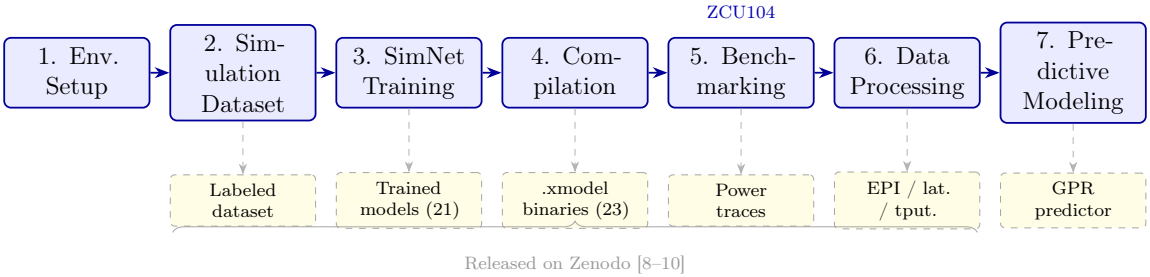

Figure 4.1: Methodology pipeline. Each stage produces a primary data artifact (dashed arrows). The three Zenodo-released datasets cover the labeled time-series simulation data, the compiled `.xmodel` binaries, and the per-batch power traces.

transfer and remote execution.

Before proceeding to data collection, the complete toolchain path was validated end-to-end. A test model was trained on the host, compiled to a `.xmodel`, transferred to the ZCU104, and executed via the VART runtime. Its output was compared against a floating-point CPU baseline. Agreement was assessed as binary classification concordance. The INT8 DPU output and the FP32 CPU output agreed on the predicted class (positive or negative anomaly) for all test windows, consistent with the expected effects of INT8 quantization on a binary classification task. Additionally, all hardware sensor paths used by the benchmark package were enumerated and verified to be readable from Python. This two-stage validation confirmed both DPU subgraph compatibility and sensor availability before any benchmarking data were collected.

4.2 Rubber Bushing Simulation

4.2.1 Physical Model

A purpose-built physics simulation generates the labeled time-series dataset on which all benchmark models are trained and evaluated. The simulation models a rubber bushing under cyclic sinusoidal displacement loading as a scalar nonlinear viscoelastic system [28] with Coulomb friction [26]. The total reaction force at each timestep is the sum of three physically distinct contributions.

Nonlinear elastic force. A linear stiffness term is combined with a cubic progressive stiffening that activates only when displacement exceeds a threshold x_{prog}:

$$F_{\text{el}} = k_0 x + k_3 \cdot \text{sign}(x) \cdot \max\big(0,\ |x| - x_{\text{prog}}\big)^3. \tag{4.1}$$

Viscoelastic force. Two parallel Maxwell branches capture fast and slow viscoelastic relaxation. Each branch state z_i evolves as

$$\dot{z}_i = k_i \dot{x} - \frac{z_i}{\tau_i}, \qquad F_{\text{ve}} = \sum_{i=1}^{2} z_i, \tag{4.2}$$

integrated with forward Euler at a timestep of $\Delta t = 1$ ms. At this rate, the stability ratio $\tau_i / \Delta t$ ranges from approximately 30 to 700 across the sampled parameter space,

ensuring numerical stability of the integration. This is a Wiechert (generalized Maxwell) model [27].

Friction force. Coulomb friction is approximated by a smooth hyperbolic tangent transition supplemented by linear viscous damping:

$$F_{\text{fr}} = F_c \cdot \tanh\left(\frac{\dot{x}}{v_s}\right) + c_v \dot{x}. \tag{4.3}$$

The prescribed excitation is a sinusoidal displacement $x(t) = \text{offset} + A\sin(2\pi f t)$ with velocity $\dot{x}$ computed analytically. The model is implemented in Python using NumPy, with no external physics library.

4.2.2 Observable Signals and Labels

Three observable sensor channels are produced per timestep. These are measured displacement x_{meas} (equal to x plus Gaussian noise, with optional low-pass filtering), measured velocity v_{meas} (numerical gradient of x_{meas}), and measured force F_{meas} (equal to the true force plus Gaussian noise, into which signal-level anomalies are injected). These three channels constitute the only inputs available to the inference model. The true internal states are not exposed.

Each timestep carries a binary anomaly label (anomaly = 1 if an anomaly is active, 0 otherwise) with a short 50 ms settling tail after each event. The simulation supports four anomaly types. These are abrupt parameter step changes (`param_step`), gradual parameter ramps (`param_ramp`), transient sensor spikes (`signal_spike`), and sensor dropouts (`signal_dropout`). Anomaly types are drawn with weighted probability, with signal-level faults occurring roughly five times more frequently than parameter-level faults.

4.2.3 Dataset Generation

The dataset consists of 3 000 independent simulation runs, organized into 100 *families* of 30 runs each. For each family, a set of base physical parameters is drawn independently and uniformly from broad prior ranges (Table A.3 in the Appendix), covering excitation amplitude and frequency, stiffness coefficients, viscoelastic relaxation times, friction parameters, and sensor noise levels. Within each family, all 30 runs share the same base parameters, with a $\pm 7\%$ perturbation applied independently to five of them (amplitude, frequency, linear stiffness, Coulomb friction, and viscous damping) to simulate natural run-to-run variability. Each run produces 30 seconds of data sampled at 1000 Hz. Family base parameters are drawn independently and uniformly at random using a fixed seed (42).

The dataset is split by family into training (70%), validation (15%), and test (15%) subsets, with each family assigned entirely to one split. Because all runs within a family share the same base parameters, no test window originates from the same operating point as any training window.

The raw time series are windowed into 512-sample windows with a stride of 256 samples. It is worth noting that within each family, the 30 runs share the same base parameters and differ only by small perturbations, meaning that training

windows from the same family cluster in feature space. This is a characteristic of the training distribution rather than a methodological flaw, but it implies that the effective diversity of the training set is closer to 70 families than to 2,100 individual runs. A `StandardScaler` is fitted on the training split only and applied to all splits. The final dataset occupies 5.1 GB in Parquet format, with each window having input shape $(3, 512)$ and a binary label derived as the maximum anomaly flag over the window timesteps.

4.3 The SimNet Model Family

4.3.1 Architectural Template

All benchmark models are instances of a custom 1D CNN architecture designed for this project, referred to as SimNet. A homogeneous model family is chosen deliberately. Varying only depth and channel width ensures that energy differences are attributable to those parameters alone. Established image classification architectures such as ResNet and MobileNet were not considered because they are 2D CNN architectures designed for spatial image data, not 1D time-series sensor signals. Existing 1D CNN families such as temporal convolutional networks or dilated residual networks were not used because they vary along multiple architectural axes simultaneously, which would prevent attributing energy differences to specific design parameters. SimNet isolates depth and channel width as the two axes of variation, providing the experimental control needed to characterize their individual and joint effects on EPI. The network takes an input tensor of shape $(B, 512, 3)$, representing batch size, timesteps, and features, matching the time-series-natural layout of the dataset. The `forward()` method permutes this internally to $(B, 3, 512)$ before passing it to the Conv1d layers, which follow PyTorch's channel-first convention. This permute is an internal implementation detail. The caller-facing interface uses the more natural time-major layout. The network then passes the permuted tensor through a sequence of convolutional blocks. Each block applies a Conv1d layer with same-padding, batch normalization, and ReLU activation. A MaxPool1d with kernel size 2 is inserted after every second convolutional block, halving the sequence length. After the final convolutional block, a chain of additional MaxPool1d(2) operations reduces the remaining feature map to length one. The output is then flattened, passed through a dropout layer, projected to a single logit by a linear layer, and squeezed to shape $(B,)$. MaxPool1d(2) is used throughout for spatial downsampling, and batch normalization is always enabled. Quantization is fixed to INT8 throughout, as produced by the Vitis AI toolchain.

Figure 4.2 illustrates the SimNet template.

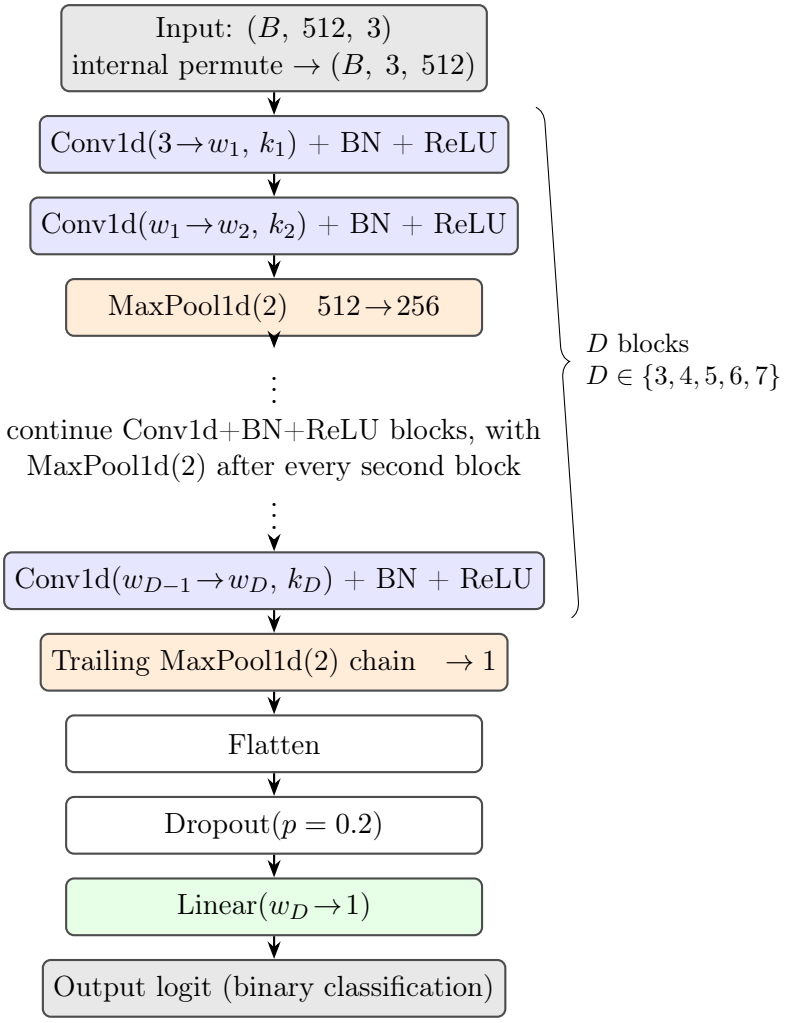

Figure 4.2: SimNet architectural template. All benchmark models are instances of this template, varying only in depth D and base channel width w_0. The convolutional kernel sizes $k_1, \ldots, k_D$ follow the depth-specific schedules listed in Table 4.1. After the convolutional blocks, additional MaxPool1d(2) operations reduce the feature map to length one. BN = batch normalization.

4.3.2 Scaling the Model Family

The benchmark suite spans more than four orders of magnitude in parameter count by varying two architectural axes while holding everything else fixed.

Depth. Five depth tiers of 3, 4, 5, 6, and 7 convolutional layers are predefined. Within each tier, the per-layer channel ratios and kernel sizes follow a fixed schedule, summarized in Table 4.1. Kernel sizes decrease from early to late layers, reflecting the common practice of using larger receptive fields in the early layers. The 24 size-scaled models are assigned to depth tiers in blocks of five (with the final tier receiving four models). The non-integer channel ratio of 3.5 in the depth-5 template is rounded to the nearest multiple of eight when instantiated, ensuring that all channel widths are integer-valued and DPU-tiling-aligned across the full base-width range.

Width. Within each depth tier, a single base channel width w is chosen by binary search to match a logarithmically spaced parameter target. The search terminates

Table 4.1: Depth tier templates. Channel ratios are relative to the base width w (rounded to the nearest multiple of 8). Kernel sizes decrease from the first to the last convolutional layer.

Depth	Channel ratios	Kernel sizes
3	[1, 2, 4]	[5, 3, 3]
4	[1, 2, 4, 4]	[9, 7, 5, 3]
5	[1, 2, 3.5, 4, 4]	[9, 7, 5, 3, 3]
6	[1, 2, 4, 4, 6, 6]	[11, 9, 7, 5, 3, 3]
7	[1, 2, 3, 4, 4, 5, 5]	[11, 9, 7, 5, 3, 3, 3]

when the achieved parameter count is within 1% of the target, and channel counts at each layer are rounded to the nearest multiple of eight to favor DPU tiling alignment. Rounding is applied after width selection, so the final parameter count may differ slightly from the target. The exact achieved values are reported in Table A.1. The 24 parameter targets are spaced logarithmically between 25 000 and 300 000 000:

$$P_i = \left\lfloor P_{\min} \cdot \left(\frac{P_{\max}}{P_{\min}} \right)^{i/23} \right\rfloor, \quad i = 0, \ldots, 23. \tag{4.4}$$

Two additional models, `model-boundary` and `model-overflow`, were constructed beyond the 24 size-scaled models to probe the contiguous memory allocator (CMA) memory and toolchain serialization limits of the platform. These are not part of the benchmark dataset. The full model family is listed in Table A.1 in Appendix A.

4.3.3 Training Procedure

Models up to and including `model-88m` (88.2 million parameters) are trained on the rubber bushing dataset. Models above this threshold use random weights. Training `model-132m` was attempted but produced severe validation F1 degradation within the first epoch, consistent with the dataset being too small to provide useful gradient signal for a model of this capacity. The decision to use random weights above 88.2 million parameters is therefore empirically motivated rather than a pre-emptive choice. These large models are included to probe the compilation and deployment limits of the toolchain rather than to contribute trained predictive performance.

All trained models use binary cross-entropy with logits as the loss function, with a per-batch positive class weight $\min(20, n_{\text{neg}}/n_{\text{pos}})$ computed from the window-level label distribution to account for class imbalance. The optimizer is AdamW [20] with a cosine annealing learning rate schedule [21] (`CosineAnnealingLR`, $T_{\max} = \text{epochs}$) and gradient clipping at every step. Training is stopped early if validation F1 fails to improve by at least a minimum delta over a patience window. Both values scale with model size. Patience increases linearly from 5 epochs (at $t = 0$) to 10 epochs (at $t = 1$), and minimum delta decreases log-linearly from 3×10^{-4} to 5×10^{-5}. The checkpoint with the best validation F1 is restored for final evaluation. The model is trained on sliding-window batches from the training split, validated on the validation split after each epoch, and evaluated on the held-out test split once training is complete.

Key hyperparameters scale with model size using a scalar $t \in [0, 1]$ defined as

$$t = \frac{\log_{10}(P) - \log_{10}(P_{\min})}{\log_{10}(P_{\max}) - \log_{10}(P_{\min})}, \quad P_{\min} = 25{,}385, \quad P_{\max} = 88{,}211{,}025, \tag{4.5}$$

where P is the model's parameter count and the bounds correspond to the smallest and largest trained models respectively. Hyperparameters interpolate linearly in t: learning rate from 2.5×10^{-3} (at $t = 0$) to 5×10^{-5} (at $t = 1$), batch size from 96 to 8, classifier dropout from 0.12 to 0.45, and block dropout from 0.03 to 0.13. AdamW weight decay interpolates from 1×10^{-4} to 6×10^{-4}, and gradient clipping norm from 0.5 to 1.0. Early stopping patience ranges from 5 to 10 epochs with a minimum improvement delta of 3×10^{-4} to 5×10^{-5}. Epoch budgets range from up to 30 epochs for models below 1 million parameters to up to 5 epochs for models between 20 and 100 million parameters. The same dataset split, feature scaler, and random seed (42) are used for all models. Model accuracy on the anomaly detection task is not an optimization target. The models are designed to cover a diverse range of architectural configurations rather than to maximize predictive performance.

4.4 Compilation Pipeline

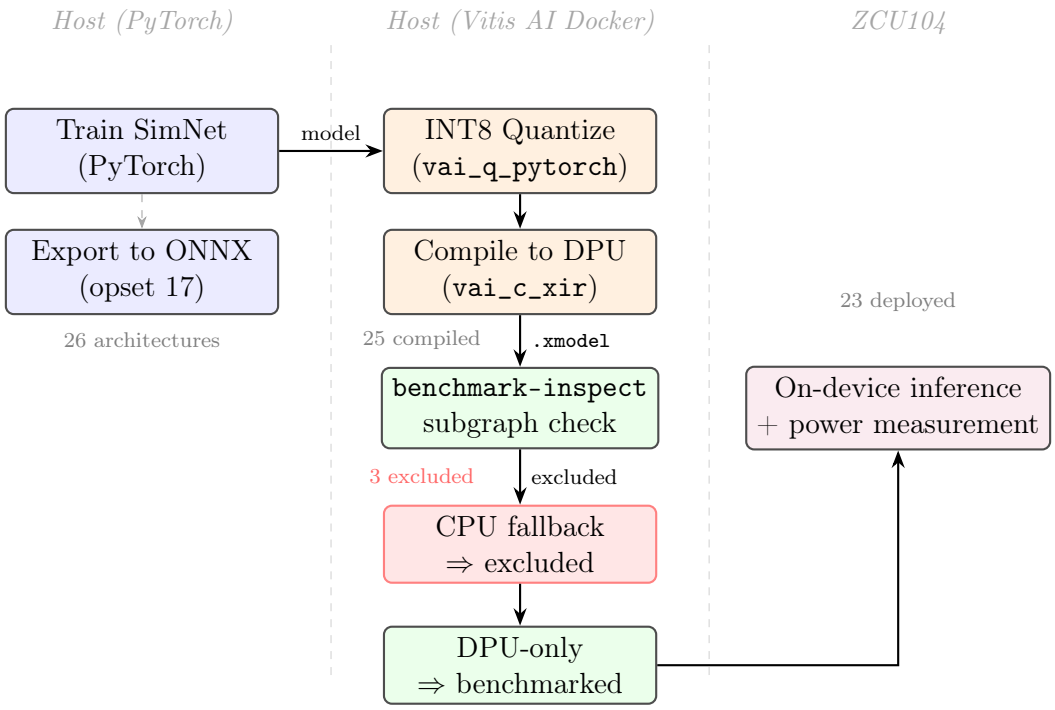

Figure 4.3: Compilation and pre-flight inspection pipeline. Models are trained in PyTorch, then quantized and compiled inside the Vitis AI Docker container. The separate ONNX export is retained as an inspection artifact and is not consumed by the quantizer. The compiled `.xmodel` files are checked by `benchmark-inspect`. Three models are excluded. One exceeds the Protocol Buffers 2 GB limit and two produce CPU fallback subgraphs.

Each model is converted to a deployable `.xmodel` binary using a deployment pipeline executed inside the Vitis AI Docker container. The first step is quantization. The Vitis AI quantizer (`vai_q_pytorch`) operates directly on the trained PyTorch model, traces it with a dummy input tensor of shape $(1, 512, 3)$, calibrates INT8 activation scale factors, and produces a quantized model graph.

An ONNX version of the floating-point model is exported separately in the host environment using opset 17, but this file is only used as an inspection artifact. It is not consumed by the quantizer. Instead, `vai_q_pytorch` internally re-exports the quantized graph to ONNX opset 17 using the PyTorch 1.13.1 exporter available inside the container. The matching host and container opset versions are therefore incidental and do not indicate that the host ONNX export is part of the quantization path.

Using a dummy input rather than real calibration data is a known simplification. Real activation distributions would, in principle, produce more representative INT8 scale factors. However, this benchmark targets energy, latency, and throughput rather than post-quantization classification accuracy. The main concern is therefore whether dummy calibration introduces a systematic EPI bias. Since INT8 MAC operation counts are determined by the architectural datapath rather than by specific input

values, calibration-induced shifts in activation scale factors are expected to affect absolute EPI values more than relative differences between models. This is a plausible assumption, but it was not empirically verified. Activation clipping behavior can vary between calibration inputs, and any resulting change in effective DPU execution could affect measured EPI. This uncertainty is therefore treated as a limitation of the study.

The second step is compilation. The `vai_c_xir` compiler maps the quantized graph to DPU instruction sequences for the ZCU104 B4096 target. For the Vitis AI compiler version used in this study, a correctly compiled model produces a computation graph with exactly three subgraphs. These consist of a USER entry point holding the input tensor descriptor, a single DPU subgraph containing all convolutional compute, and a trailing CPU wrapper that performs only output dequantization. Any additional CPU subgraph containing convolutional or arithmetic compute indicates that the compiler could not map those operations onto the B4096 instruction set architecture (ISA) and instead fell back to Advanced RISC Machine (ARM) CPU execution.

CPU fallback invalidates energy and latency measurements. CPU-side convolutional compute draws current from the processing system (PS) power rails alongside the DPU, which contaminates the energy reading in a model-dependent way that cannot be separated after measurement. In addition, each DPU–CPU–DPU subgraph boundary introduces extra latency through DDR data transfers and cache cold-start effects on the ARM cores.

To detect such fallbacks before measurement begins, all compiled `.xmodel` files are inspected with `benchmark-inspect`, a custom pre-flight tool developed as part of this project. The tool enumerates the subgraph layout of each compiled graph and classifies non-DPU subgraphs as either benign wrappers or CPU fallbacks. Benign wrappers are accepted only when they perform single-operator buffer management or output handling rather than model compute.

Of the 26 models in the family, 25 compile successfully. `model-overflow` fails because the serialized XIR graph exceeds the Protocol Buffers 2 GB hard limit. Of the 25 compiled models, `model-300m` and `model-boundary` fail the pre-flight inspection because the compiler fragments their graphs into alternating DPU and CPU subgraphs at large channel widths. Both models are excluded from the benchmark dataset, leaving 23 models for measurement (Table A.1).

4.5 Benchmarking Pipeline

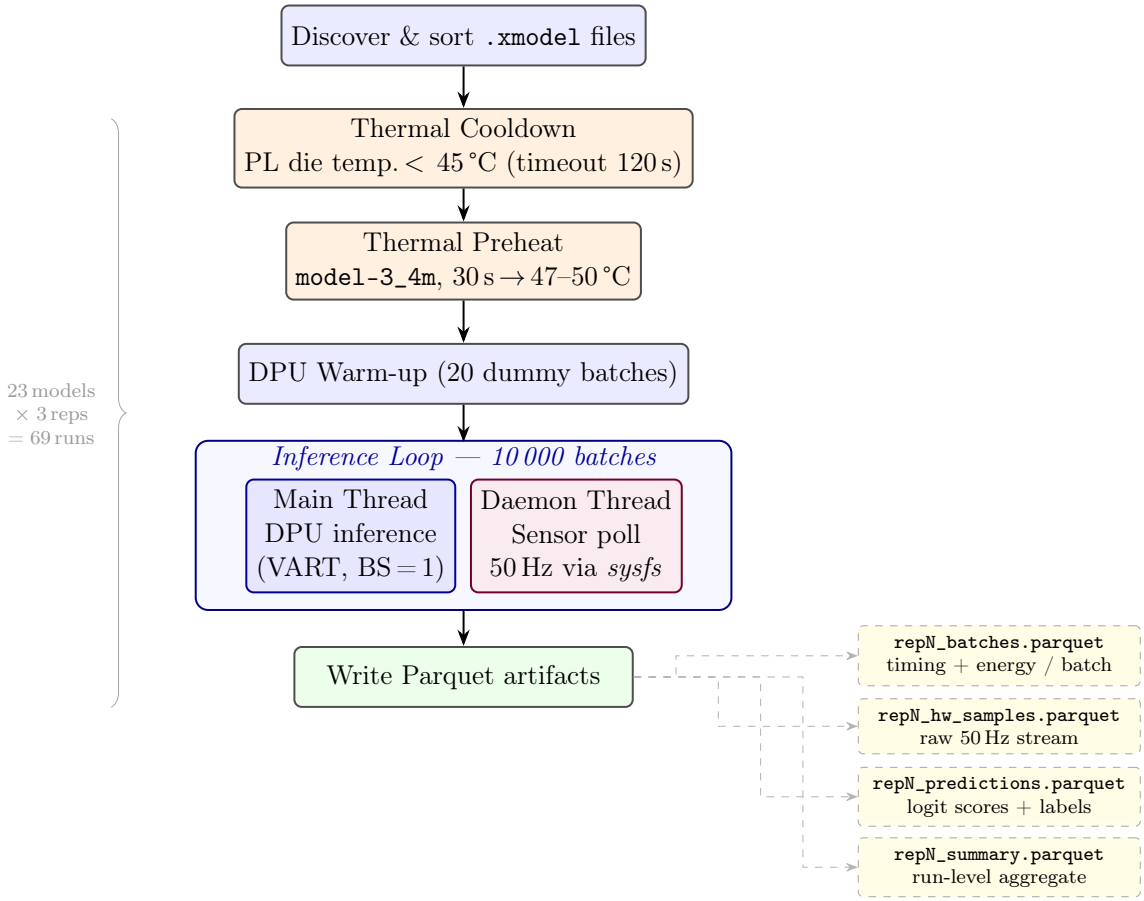

Figure 4.4: Benchmark pipeline for a single repetition. Orange steps implement the two-phase thermal pre-conditioning protocol (cooldown and preheat) applied before every repetition without exception. During the inference loop, the main thread drives DPU inference via VART while a daemon thread concurrently polls 76 hardware sensor channels at 50 Hz via *sysfs*. Each repetition produces four Parquet files (dashed arrows). The full campaign runs the discover step once and then executes 23 models in three independent repetitions each (69 runs total).

4.5.1 Package Architecture

The benchmarking pipeline is a custom Python package, developed as part of this project and deployed on the ZCU104. DPU inference runs on the programmable logic fabric in parallel with hardware sensor polling on the ARM CPU, producing per-batch timing, power, and energy measurements alongside model predictions. The package is structured as a single-process, two-thread application where the main thread executes the inference loop while a daemon thread polls sensors at 50 Hz.

Each of the 23 models is measured in three independent repetitions of 10 000 inference samples each. The choice of three repetitions reflects a deliberate trade-off that provides sufficient between-run variance estimation (the minimum for a sample standard deviation with any degrees of freedom) while keeping the total campaign

at 69 sessions. The binding constraint on predictive modeling accuracy is $N = 23$ models, not the number of repetitions per model. Increasing repetitions beyond three would not improve the predictive model and would have been better spent extending the model family if additional deployable configurations existed. Three repetitions was therefore the appropriate allocation given the platform constraints. This sample count also provides a statistically stable within-repetition EPI estimate. To ensure fair and consistent comparison across models, a thermal conditioning protocol is applied before every repetition, with the board first cooled below a fixed temperature threshold, then brought to a consistent operating temperature by a pre-heating workload. This guarantees that every measurement begins from the same thermal baseline regardless of which model was benchmarked previously. The same 10 000 input windows are presented in the same order for every repetition of every model, ensuring that variability in energy and latency is attributable to the hardware and the model architecture rather than to differences in the input distribution. A single CLI invocation runs the full sweep of 69 repetitions automatically. If a run is interrupted, incomplete artifacts are detected and discarded on the next invocation, so only complete repetitions enter the dataset. Figure 4.4 illustrates the per-repetition benchmark pipeline and the artifacts it produces.

4.5.2 Inference Timing and Sensor Sampling

Duration measurements used Python's `time.perf_counter()`, a monotonic high-resolution timer, to avoid errors from system wall-clock adjustments during benchmark execution. All host-side pre- and post-processing occurred outside this timed window. All compiled models had a fixed DPU batch size of one, so one batch equaled one inference and the timed interval equaled per-inference latency directly.

The hardware monitor polls 76 sensor channels at 50 Hz in a background thread independent of the inference cadence. Channels include five temperature sensors, 12 power rails with associated voltages and currents from the IRPS5401 integrated power system and INA226 current/power monitor, 29 on-die analog mixed-signal (AMS) voltage channels, and four per-core CPU frequency readings. Sensor paths are discovered dynamically at startup.

Table 4.2: Key sensor channels recorded during benchmarking on the ZCU104. All channels are sampled at 50 Hz via the `sysfs` interface. EPI is derived from `pout1`, which covers the full PS/PL power domain.

Channel	Description	Unit	Source
`pout1`	Total PS/PL output power	W	IRPS5401 PMIC
`pout2`	SoC core supply power	W	IRPS5401 PMIC
`pout3`	PS/PL auxiliary power	W	IRPS5401 PMIC
`vout1`–`vout3`	Associated rail voltages	V	IRPS5401 PMIC
Temp. 1–5	Die and board temperatures	°C	On-chip / board
AMS channels	29 on-die voltage monitors	V	Zynq AMS
CPU freq.	Per-core frequency readings	MHz	`cpufreq`

Derived metrics. Latency = wall-clock duration of one inference batch (ms). Throughput = reciprocal of mean batch latency (inf/s). EPI = `pout1` integrated over one inference batch using the trapezoidal rule (μJ).

For each inference batch, sensor samples captured within the batch's wall-clock window are used to compute per-batch energy via trapezoidal integration:

$$E_{\text{batch}} = \int_{t_{\text{start}}}^{t_{\text{end}}} P(t)\,\mathrm{d}t \approx \sum_{k} \frac{P_k + P_{k+1}}{2}\,\Delta t_k. \tag{4.6}$$

For batches where inference completes faster than the 20 ms polling interval (which occurs for small models whose per-batch latency falls below this threshold), the nearest available sensor sample is used with a power × duration approximation instead. The per-batch column `hw_n_samples` records how many sensor samples were captured per window, providing a per-batch indicator of which energy computation method was used. For the smallest models (depth tier 3), this fallback path applies to 100% of batches. For depth tier 4 models it applies to the majority, and for models above approximately 38 million parameters (from `model-39m` onward) the trapezoidal method is used for all batches. This breakdown is examined during data processing and is reported as a per-model fallback rate in the Results chapter.

Before each repetition's measured inference begins, 20 dummy batches of zero-filled input are executed to bring the DPU pipeline to steady-state occupancy. This count was established by observing per-batch latency during an initial calibration run: latency stabilizes within the first 10–15 batches for all models in the family, so 20 provides a conservative margin. These warm-up batches are not recorded.

4.5.3 Thermal Pre-Conditioning

To ensure every model is measured from a consistent thermal baseline, a two-phase conditioning protocol is applied before every repetition without exception, including between consecutive repetitions of the same model.

In the cooldown phase, the programmable logic (PL) die temperature is polled at 1 Hz via a direct sysfs read. The system waits until this reading falls below 45°C, with a 120-second timeout. The 45°C threshold was chosen because it sits comfortably above the ambient board temperature (approximately 35–38°C at rest) while remaining reachable within a reasonable cooldown period. The subsequent pre-heat phase drives the board to 47–50°C, so the 45°C threshold guarantees that every pre-heat starts from a consistent below-target temperature. If the timeout is reached, a warning is logged and the pre-heat phase proceeds regardless. In practice the timeout was never triggered across all 69 repetitions. Cooldown durations ranged from approximately 9 to 60 seconds depending on the thermal residue of the preceding run.

In the pre-heat phase, a 30-second workload runs continuous DPU inference using `model-3_4m` as a reference model. This model was selected because it produces a moderate and stable power draw during DPU inference, large enough to warm the PL fabric to a representative operating temperature but small enough that 30 seconds of continuous inference completes several hundred batches, allowing the board temperature to stabilize fully before measurement begins. At approximately 3.4 million parameters it is the 13th of 23 models by parameter count and sits near the median by compiled file size, making it a reasonable proxy for average DPU utilization. Zero-filled input tensors are used because DPU compute-path

latency and switching activity on INT8-quantized convolutions are assumed to be approximately data-independent for fixed weights, so a constant input is expected to produce a representative thermal load without the cost of dataset I/O on the memory-constrained device. The PL temperature at the end of the pre-heat phase is recorded as the `preheat_temp_C` field in the repetition summary. Across all 69 repetitions it ranged from approximately 47°C to 50°C, confirming consistent starting thermal conditions.

4.5.4 Run Design and Output

Each of the 23 models is measured in three independent repetitions of 10 000 inference samples each. Models are benchmarked in ascending order of compiled file size, so that smaller and faster models are validated first. Since the conditioning protocol resets thermal state before every run, this ordering does not introduce thermal bias.

Each repetition produces four Parquet files. `repN_batches.parquet` contains one row per batch, recording inference timing, per-rail power and integrated energy, the number of hardware samples captured in the window, and mean board temperatures. Per-rail energy is pre-integrated at write time, so rail selection during analysis requires only a column projection rather than a recomputation from raw samples. `repN_hw_samples.parquet` contains the raw 50 Hz sensor stream with each row tagged to its `batch_id`. `repN_predictions.parquet` records labels and logit scores per batch. `repN_summary.parquet` contains one-row run-level aggregates including a preliminary whole-board energy-per-inference estimate. This serves as a scaling indicator only, and the authoritative EPI is derived in the following stage from the appropriate power rail.

4.6 Data Processing

The raw benchmark output, comprising 690 000 batch rows across 23 models and 3 repetitions, is processed by a cleaning pipeline that produces a 23-row modeling-ready dataset.

Rail selection. The `pout1` output rail of the IRPS5401 power management controller is selected as the primary EPI rail. It covers the main PS/PL power domain, which includes the DPU fabric. The ARM processor cores are fixed at 1.2 GHz throughout all benchmark runs, confirmed by the constant CPU frequency readings in the hardware monitor output. While fixed frequency reduces ARM dynamic power variation, it does not eliminate it entirely. Instruction throughput and cache activity still vary with inference duration. Cross-model variation in `pout1` is therefore primarily, but not exclusively, attributable to the DPU workload. This limitation is discussed further in Section 6.10. The full-board INA226 input rail (approximately 34 W during inference) is not used because it conflates DDR memory, Ethernet, and other board-level consumers unrelated to DPU inference. No single rail on the ZCU104 is purely PL-isolated. This is a hardware limitation of the evaluation platform and is stated transparently as such.

EPI estimation. Per-batch EPI in microjoules is estimated from the pre-integrated per-batch `pout1` energy written by the hardware monitor. The per-repetition EPI

estimate is the arithmetic mean of the 10 000 per-batch values. The final per-model EPI is the unweighted mean of the three per-repetition estimates.

$$\widehat{\mathrm{EPI}}_m = \frac{1}{R}\sum_{r=1}^{R}\hat{\mu}_{m,r}, \quad R = 3, \tag{4.7}$$

with measurement uncertainty expressed as the sample standard deviation across repetitions (Bessel-corrected). This two-level design is deliberate. Within-repetition variance captures sensor quantization noise and batch-to-batch variation within a single thermal realization, while across-repetition variance additionally captures run-to-run variability from thermal drift and VART runtime reload state. The across-repetition standard deviation is therefore the more conservative and methodologically appropriate uncertainty source for a per-model characterization.

Outlier detection. An automated check flags any repetition whose EPI deviates more than two within-model standard deviations from the model mean across repetitions. A threshold of two standard deviations is chosen as a conservative criterion that catches genuine outliers while tolerating the natural run-to-run variance expected from thermal and scheduling noise. Flagged repetitions are annotated but not automatically excluded, allowing the reported standard deviation to honestly reflect measurement spread rather than masking it through silent removal. It should be noted that for a sample of exactly three repetitions the maximum possible absolute deviation of any observation from the sample mean, expressed in units of the sample standard deviation, is $2/\sqrt{3} \approx 1.15$. The 2σ threshold is therefore impossible to trigger by construction at $n = 3$. This check is therefore non-operative in this study and provides no empirical information about data quality here. It is retained as infrastructure for future campaigns with $n > 3$ where the threshold becomes triggerable.

Feature engineering. The cleaned dataset is joined with architecture features derived from model configuration (architecture-derived features) and from the compiled `.xmodel` artifact (one post-compilation graph feature). The majority are computable before hardware benchmarking begins. The candidate pool comprises 29 features organized into five conceptual groups. These are log-transformed scale features (`log10_params`, `log10_macs`, `log10_width`, and derived variants), parameter and compute density features (`params_per_layer`, `mac_per_param`, `macs_per_dpu_op`), pooling and sequence-length features (`n_regular_pool_ops`, `final_seq_len_before_trailing`), kernel geometry features (`mean_kernel_size`, `receptive_field`), and channel progression features (`channel_growth_ratio`, `total_activations`). One feature, `dpu_op_count`, is obtained from the compiled `.xmodel` rather than from the model configuration alone, and therefore requires the Vitis AI compilation step as a prerequisite for prediction. Since compilation is a prerequisite for deployment regardless, this does not add a new practical burden, but it means the predictor cannot be applied to a model that has not yet been compiled. The interaction term `depth_x_log_width` $= D \times \log_{10}(w)$ is included as a candidate capturing the joint effect of depth and channel scale. Complete definitions for all 29 features are provided in Table A.2 in the Appendix. No measured hardware quantities are used as predictors.

Total MACs are computed analytically from the per-layer channel widths and

kernel sizes, accounting for the sequence-length halving at each pooling step:

$$\text{MACs} = \sum_{i=1}^{D} C_{\text{in},i} \cdot C_{\text{out},i} \cdot k_i \cdot L_i, \quad L_i = \frac{512}{2^{\lfloor (i-1)/2 \rfloor}}. \tag{4.8}$$

The trailing MaxPool chain contributes zero MACs and is excluded. This formula was verified against the stored values in the model configuration file for all 23 models.

Exploratory analysis. To address RQ2, the relationships between architecture parameters and measured EPI, latency, and throughput are characterized through a combination of scaling analyses and correlation analysis. RQ2 is answered descriptively rather than confirmatorily, with no specific hypotheses about which architectural parameters dominate are pre-registered, and all findings are patterns identified in the data rather than outcomes of hypothesis tests. Log-log regression of each target against parameter count quantifies the power-law scaling behavior across the model family. The Roofline model [29] places each model's achieved compute throughput against the theoretical B4096 compute and memory bandwidth ceilings. Per-model power scaling and repetition consistency are examined to characterize measurement stability and to separate power effects from latency effects in the EPI signal. Univariate Spearman rank correlations between each of the 29 candidate features and each target are computed to identify which architectural properties are most strongly associated with on-device performance, independently of any modeling assumptions.

4.7 Predictive Modeling

Terminology note. Throughout this section, *model* refers to the statistical predictive model (GPR), not to the SimNet CNN architectures being characterized. The terms *SimNet model* and *GPR predictor* are used where disambiguation is needed.

The predictive modeling stage fits regressors that map architecture parameters to deployment-level EPI, latency, and throughput without requiring hardware access, directly addressing RQ1 and RQ3. Three targets are modeled independently using the same pipeline. The dataset comprises $N = 23$ models, which is the complete set satisfying all deployment constraints simultaneously (single DPU subgraph, passing pre-flight inspection, and covering the intended parameter range). This figure represents the population of deployable SimNet architectures on this hardware configuration rather than a sample from a larger available pool. Additional data points would require changing the model family, the hardware platform, or the toolchain version. No models within the defined family were excluded or withheld. Leave-one-out cross-validation (LOO-CV) is used throughout because it maximizes the training set size for each evaluation fold at this sample size. Gaussian process regression (GPR) [22] is selected as the primary *predictive model* (distinct from the SimNet CNN models being characterized). Unlike parametric models that fit a fixed-form equation to the data, GPR places a probability distribution over all functions that could plausibly explain the observations. Given a new architecture, GPR returns not just a point prediction but a full predictive distribution, from which both a mean estimate and a prediction interval are derived. This probabilistic output is essential here. A wide interval signals that the query architecture lies far from the training data and that hardware measurement is advisable before committing to that design.

GPR is particularly well-suited to small datasets. With $N = 23$ training points, flexible parametric models such as polynomial regression have enough degrees of freedom to fit the training data closely while producing unreliable predictions between those points. GPR mitigates this risk because its uncertainty grows automatically in regions where training data is sparse, rather than committing to a confident wrong answer. The kernel function determines how similarity between architectures is measured. The ARD-RBF kernel used here assigns a separate length-scale parameter to each input feature, so the model can learn which features are most predictive by shrinking the length-scale of uninformative dimensions during training.

Feature pre-selection. The 29 candidate architecture features are highly collinear because the SimNet family varies along a tightly parameterized design space of depth and channel width. Variance inflation factor elimination is not applied, as it would collapse the candidate set to a single feature on this dataset. Instead, hierarchical clustering on pairwise Spearman rank distance groups features with a minimum mutual correlation of $|\rho| \geq 0.85$ into clusters, and one representative per cluster is retained per target, selected as the cluster member with the largest absolute Spearman correlation to that target. This procedure consistently yields four clusters across all three targets. The result is a dominant size-and-compute cluster, a pooling-and-kernel-geometry cluster, and two singletons (mean kernel size and arithmetic intensity), giving four cluster-representative candidate features per target. In addition, the engineered interaction term `depth_x_log_width` ($= D \times \log_{10} w$) was evaluated as a fifth candidate alongside the cluster representatives. Note that `dpu_op_count` satisfies $40+8D$ on this platform, making it a linear rescaling of depth. It carries no additional information beyond `depth` within this family, and would only become independently informative on a platform where the DPU graph structure varies non-monotonically with depth. Constructed from two members of Cluster 1 (depth and log-width), it is a manual addition outside the systematic clustering framework. It was included because no single cluster member encodes their joint effect.

Forward feature selection. A greedy forward selector evaluates each cluster representative by fitting a Gaussian process regressor under leave-one-out cross-validation on the full $N = 23$ dataset, using mean absolute percentage error (MAPE) on the leave-one-out (LOO) predictions as the criterion. The candidate that yields the lowest MAPE is accepted only if it strictly reduces MAPE relative to the current best. Otherwise, selection terminates. The maximum number of selected features is capped at $\lfloor\sqrt{N}\rfloor = 4$. A Gaussian process regressor is used as the selection scorer because it is the most flexible candidate and therefore provides the most conservative assessment of whether a feature carries independent predictive signal under LOO. It should be noted that this procedure is partially nested, as feature selection evaluates each candidate using all $N = 23$ labels, so the reported LOO-CV metrics in the Results chapter are lower bounds on what a fully nested protocol would produce. This is an acknowledged limitation of the evaluation protocol. At $N = 23$, a fully nested alternative would train on only 22 points per outer fold and evaluate one candidate at a time, producing highly variable feature selections with negligible benefit over the partial approach. The four-cluster bottleneck further limits the candidate set to four features, making the selection decision robust to small perturbations of the training set. Reported LOO-CV MAPEs are stated as lower bounds throughout the Results chapter.

Model comparison. Once features are fixed per target, five candidate estimators are compared under the same outer LOO loop. The naive mean baseline predicts the training mean for every query and serves as a lower bound on useful accuracy. Its percentage errors exceed 2400% because the targets span several orders of magnitude. A mean prediction severely overestimates the smallest models and underestimates the largest models, producing large relative errors at the extremes. Linear regression and Ridge regression fit a weighted sum of the input features and are included because they are fast and interpretable, but are constrained to linear relationships. Degree-2 polynomial Ridge regression can capture nonlinear trends but with $N = 23$ it has enough degrees of freedom to overfit the training points while extrapolating poorly to the sparse depth-6 and depth-7 region. GPR with an anisotropic ARD-RBF kernel and an additive noise term is the most flexible candidate and the only one that provides model-based prediction intervals. All features are standardized to zero mean and unit variance before fitting. For each (target, model) combination, MAPE, log-MAPE (MAPE computed on log-transformed predictions and actuals), median absolute percentage error, R^2, and Spearman rank correlation are reported. The best estimator per target is selected by LOO MAPE.

A depth-stratified secondary evaluation additionally assesses whether models generalize within individual depth tiers. For each tier and target, the models in that tier (five for depths 3–6, three for depth 7 after exclusions) are evaluated under LOO-CV with the globally selected features and the GPR estimator. For depth 7 specifically, LOO-CV trains on two models and tests on one, which provides almost no basis for generalization claims. The depth-7 within-tier results are therefore reported for completeness only and should not be over-interpreted. This directly tests within-tier interpolation performance, which is the practically relevant question of whether EPI and latency can be predicted for a new model at a known depth. An EPI regime analysis further tests whether treating the depth-3 models as a separate operating regime improves prediction accuracy for the remaining 18 models. This analysis is explicitly exploratory, as the depth-3 boundary was identified after observing anomalous power behavior in the data rather than from a theoretical prior. Any reported accuracy improvement from the regime split is in-sample and should not be interpreted as a validated finding without independent replication on a different dataset or hardware platform.

The feature selection procedure produces the following final feature subsets, which are used in all subsequent model comparisons and evaluations. EPI is modeled using the depth-width interaction term `depth_x_log_width` ($= D \times \log_{10} w$, the product of depth and the log of base channel width). Latency is modeled using log-transformed parameter count alone. Throughput is modeled using `depth_x_log_width` and mean kernel size as a second feature. The rationale for and implications of these selections are discussed in the Results chapter.

Chapter 5

Results and Analysis

This chapter reports the empirical results of the benchmarking and predictive modeling pipeline, organized into eight sections: (1) classification task accuracy and dataset composition, establishing the 23-model deployable set; (2) compiler deployability constraints, documenting the three excluded architectures and the primary compiler finding; (3) measurement quality and repeatability, confirming thermal stability and low run-to-run variance; (4) EPI sensor coverage, characterizing the nearest-sample fallback rate as a step function of model size; (5) scaling laws, presenting power-law relationships for EPI, latency, throughput, and power across four orders of magnitude in parameter count; (6) roofline analysis, placing all benchmarked models against the B4096 theoretical ceilings; (7) feature correlation analysis, identifying which architecture features drive performance; and (8) predictive modeling results, reporting GPR accuracy, estimator comparison, depth-stratified evaluation, and the exploratory EPI regime analysis. All reported values are means over three independent repetitions unless noted otherwise, and all EPI values are derived from the `pout1` power rail of the IRPS5401 controller, as described in Section 4.6. Throughout this chapter, measured results are plotted with points colored by depth tier (number of convolutional layers, $D \in \{3, 4, 5, 6, 7\}$). This coloring separates the two design axes of the SimNet family (parameter count and depth) and makes it immediately visible whether depth carries explanatory power beyond parameter count alone.

Table 5.1: Main empirical findings in the Results and Analysis chapter.

Topic	**Finding**
Deployable model set	23 of 26 planned SimNet architectures were benchmarked successfully. Three were excluded due to compilation failure or CPU fallback.
Compiler deployability	`model-200m` compiled as a single DPU subgraph, while the structurally identical but wider `model-300m` produced CPU fallback.
Measurement stability	Thermal pre-conditioning produced starting temperatures of 47.8–50.2°C, and 22 of 23 models had below 1% EPI coefficient of variation across repetitions.
EPI estimation regimes	Fast models used the nearest-sample EPI approximation, while models from `model-39m` onward used fully trapezoidal integration.
EPI scaling	EPI follows a power law, $E \propto P^{0.749}$, with $R^2 = 0.984$.
Latency scaling	Latency follows a sublinear power law with parameter count: $\log_{10} T = 0.755 \log_{10} P - 3.689$, with $R^2 = 0.981$.
Power plateau	Mean `pout1` power rises from approximately 5.5–5.9 W for depth-3 models to approximately 7.8–8.3 W for depth-6 and depth-7 models.
Roofline result	All models have enough arithmetic intensity to avoid being memory-bound, but they still achieve only a fraction of the B4096 compute peak due to poor DPU utilization.
Feature selection	EPI was modeled using `depth_x_log_width`, latency using `log10_params`, and throughput using both `depth_x_log_width` and `mean_kernel_size`.
Prediction accuracy	GPR achieved lower-bound MdAPEs of 13.8%, 12.6%, and 15.4% for EPI, latency, and throughput respectively, with 95.7% empirical coverage for EPI prediction intervals.

5.1 Accuracy and Deployed Models

The full SimNet design comprised 26 architectures. Three did not proceed to hardware benchmarking (Section 5.2), leaving 23 deployable models as the basis for all subsequent measurements. These 23 models span from 25K to 200M parameters across five depth tiers of three to seven convolutional layers.

F1 scores on the rubber bushing anomaly detection task range from 0.848 to 0.874 across the 21 trained models. `model-132m` and `model-200m` use randomly initialized weights and do not contribute to this range. The smallest trained model (`model-25k`, 25K parameters) achieves a score within this range, indicating that the classification task is capacity-saturated at the smallest architecture tested. Among the 21 trained models, no monotone trend between parameter count and F1 is observed. Because all 21 trained models achieve comparable accuracy, classification performance is not

a prediction target in the subsequent analysis. The 21 trained models are treated as interchangeable in classification quality for the purposes of benchmarking since the study focuses on EPI, latency, and throughput.

Table 5.2 reports per-model training and classification metrics. Test F1 scores range from 0.848 (`model-58m`, 4 training epochs) to 0.874 (`model-198k`), a spread of approximately 2.6 percentage points, confirming that architecture size does not materially affect classification quality within this family.

Table 5.2: Per-model SimNet training and classification metrics on the held-out test set ($n = 52{,}200$ windows). Models from `model-132m` onward use randomly initialized weights and were not trained. Classification metrics are intentionally absent for these models.

Model	Params	D	w_0	MACs	Epochs	Test F1	PR-AUC	ROC-AUC
model-25k	25,385	3	32	1.56×10^7	19	0.869	0.875	0.902
model-39k	39,385	3	32	2.35×10^7	19	0.871	0.877	0.904
model-59k	58,713	3	40	3.52×10^7	18	0.869	0.877	0.903
model-85k	84,521	3	56	5.20×10^7	18	0.870	0.877	0.903
model-129k	128,545	3	64	7.81×10^7	16	0.866	0.872	0.901
model-198k	197,705	4	40	1.14×10^8	17	0.874	0.876	0.903
model-282k	282,169	4	56	1.65×10^8	14	0.866	0.875	0.902
model-440k	439,545	4	64	2.54×10^8	16	0.873	0.875	0.900
model-658k	657,921	4	80	3.82×10^8	12	0.866	0.873	0.899
model-973k	972,537	4	96	5.63×10^8	14	0.866	0.874	0.899
model-1_5m	1,473,537	5	104	7.03×10^8	14	0.867	0.874	0.901
model-2_2m	2,229,361	5	128	1.06×10^9	10	0.863	0.871	0.897
model-3_4m	3,378,601	5	152	1.60×10^9	11	0.868	0.872	0.899
model-5_1m	5,051,921	5	192	2.41×10^9	10	0.862	0.870	0.897
model-7_6m	7,583,625	5	232	3.59×10^9	8	0.863	0.871	0.899
model-11m	11,398,681	6	184	4.57×10^9	7	0.863	0.868	0.897
model-17m	17,283,305	6	224	6.94×10^9	5	0.857	0.867	0.895
model-26m	25,840,833	6	280	1.04×10^{10}	5	0.866	0.871	0.900
model-39m	38,802,553	6	344	1.56×10^{10}	4	0.860	0.873	0.903
model-58m	58,406,369	6	416	2.34×10^{10}	4	0.848	0.869	0.897
model-88m	88,211,025	7	536	3.14×10^{10}	3	0.862	0.870	0.900
model-132m	132,307,793	7	664	4.71×10^{10}	—	—	—	—
model-200m	199,565,105	7	808	7.10×10^{10}	—	—	—	—

5.2 Compiler Deployability

Of the 26 planned architectures, three were excluded before benchmarking. `model-overflow` produced a compiled `.xmodel` binary that exceeded the 2 GB Protocol Buffers size limit. Both `model-boundary` and `model-300m` compiled without size errors but failed the pre-flight subgraph inspection because parts of their computation were assigned to the ARM CPU rather than the DPU. This condition is termed CPU fallback. Mixing DPU and CPU execution within the same inference invalidates direct EPI and latency comparisons with the rest of the dataset, so both models were excluded.

The CPU fallback in `model-300m` is the most informative of these exclusions and constitutes a primary finding of this study. Figure 5.1 shows the relevant inspection output. `model-200m` compiles to the expected three-subgraph structure: one USER wrapper, one DPU compute subgraph, and one CPU wrapper for output handling. In contrast, `model-300m` compiles into seven subgraphs, with three CPU fallback regions interleaved between DPU regions. These CPU fallback regions contain operators such as `depthwise-conv2d-fix`, `conv2d-fix`, `reshape-fix`, and `fix2float`, indicating that model compute has escaped the DPU-only execution path.

```
xilinx@pynq:~/benchmark benchmark-inspect
[OK] model-200m
  Subgraphs: 3  (DPU=1, wrappers=2, fallback=0)
  [0] USER (wrapper): 1 ops
  [1] DPU: 80 ops
  [2] CPU (wrapper): 1 ops
[FAIL] model-300m
  Subgraphs: 7  (DPU=3, wrappers=1, fallback=3)
  [0] USER (wrapper): 1 ops
  [1] DPU: 51 ops
  [2] CPU (fallback): 3 ops
        * const-fix                ×2
        * depthwise-conv2d-fix     ×1
  [3] DPU: 5 ops
  [4] CPU (fallback): 7 ops
        * const-fix                ×4
        * depthwise-conv2d-fix     ×1
          reshape-fix              ×1
        * conv2d-fix               ×1
  [5] DPU: 14 ops
  [6] CPU (fallback): 5 ops
        * const-fix                ×2
          fix2float                ×1
        * conv2d-fix               ×1
          reshape-fix              ×1
```

Figure 5.1: Pre-flight subgraph inspection for `model-200m` and `model-300m`. `model-200m` passes with one DPU compute subgraph and two benign wrappers. `model-300m` fails because the compiled graph is fragmented into alternating DPU and CPU fallback subgraphs. The fallback regions contain compute operators, including depthwise and standard convolution operators, so the model is not DPU-only.

This finding is significant because the SimNet architecture uses only standard `Conv1d` operations. No depthwise or separable convolutions appear anywhere in the model design. The compiled binary for `model-300m`, however, contains depthwise-separable and 2D convolution operators that were not present in the original PyTorch architecture. The most direct interpretation is that the Vitis AI compiler represented some standard 1D convolutions through an equivalent operator decomposition at the large channel widths used by `model-300m`. Parts of this decomposed graph fall outside the DPUCZDX8G instruction set architecture (ISA) support boundary, causing partial CPU fallback.

Crucially, `model-200m` uses the same depth-7 architectural template as `model-300m`, but with a base channel width of 808 instead of 992. `model-200m` compiles without

CPU fallback to a single DPU compute subgraph. The two models are therefore structurally identical and differ only in channel scale, placing the deployability threshold between base widths 808 and 992 for depth-7 architectures. The inferred channel-width threshold at which this decomposition becomes ISA-incompatible is not documented as a user-facing constraint, and no mechanism in the standard Vitis AI workflow can predict this outcome without attempting compilation. The implications of this finding for deployment practice are discussed in Section 6.7.

5.3 Measurement Quality

The thermal conditioning protocol successfully controlled starting temperatures across all 69 repetitions. Preheat temperatures at the start of each run ranged from 47.8°C to 50.2°C, a spread of 2.4°C over the full campaign. This narrow range confirms that the 30-second preheat on model-3_4m produces a repeatable thermal starting point across all models.

Figure 5.2 shows per-repetition EPI for all 23 models. The coefficient of variation (CV) across repetitions remains below 1% for 22 of the 23 models (`model-2_2m` is the exception at CV $\approx 1.4\%$). No repetition from any model exceeded the 2σ automated flagging threshold, and no repetitions were excluded on quality grounds. It should be noted that for a sample of exactly three repetitions, the maximum absolute deviation of any observation from the mean is bounded by $2/\sqrt{3} \approx 1.15\sigma$, so the 2σ criterion cannot be triggered by construction for $n = 3$ and provides no information about data quality in this study beyond confirming measurements are internally consistent.

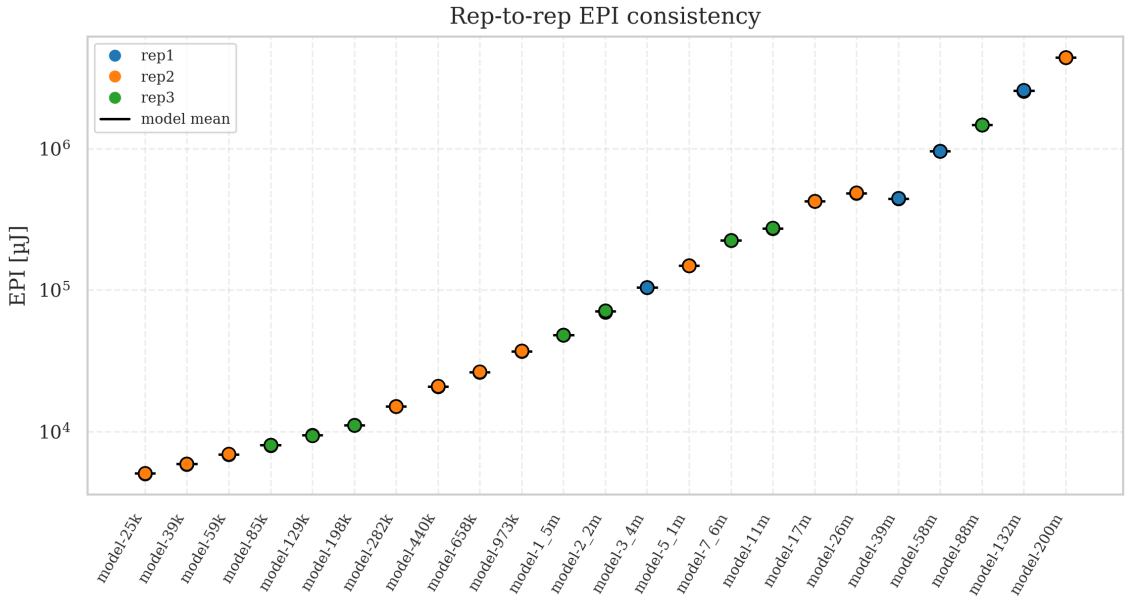

Figure 5.2: Per-repetition EPI for all 23 models sorted by model size, plotted on a log scale. Colored markers show repetitions 1, 2, and 3. Horizontal bars mark the per-model mean. The tight grouping of markers across all models confirms low run-to-run variability.

The high repeatability across repetitions, combined with consistent thermal

conditions, indicates that measured EPI differences between models are unlikely to be dominated by run-to-run noise. This consistency is the foundation on which all subsequent cross-model comparisons rest, subject to the measurement-regime caveats discussed below.

5.4 EPI Estimation Regimes

The hardware monitor polls the `pout1` rail at a nominal 50 Hz rate (20 ms intervals). When inference duration exceeds one polling interval, the trapezoidal rule integrates timestamped samples to produce the per-batch EPI estimate. When inference is shorter than one polling interval, no interior samples are captured, and the estimator falls back to a nearest-sample power approximation (Section 4.6). The fallback rate, defined as the fraction of batches using the nearest-sample approximation, characterizes how accurately the integrator resolves the power profile for each model.

Figure 5.3 shows the fallback rate for each model as a function of parameter count. The result is a near-perfect step function. All depth-3 models (0.81–1.49 ms median latency), all depth-4 models (1.72–5.19 ms), all depth-5 models (6.4–27.8 ms), and model-11m (depth-6, 11.4M parameters, 34.6 ms) operate at essentially 100% fallback. The single transitional case is model-26m (depth-6, 25.8M parameters, 78.8 ms) at 56.3% fallback. All models from model-39m (depth-6, 38.8M parameters, 119 ms) onward operate at 0% fallback and use fully trapezoidal EPI integration.

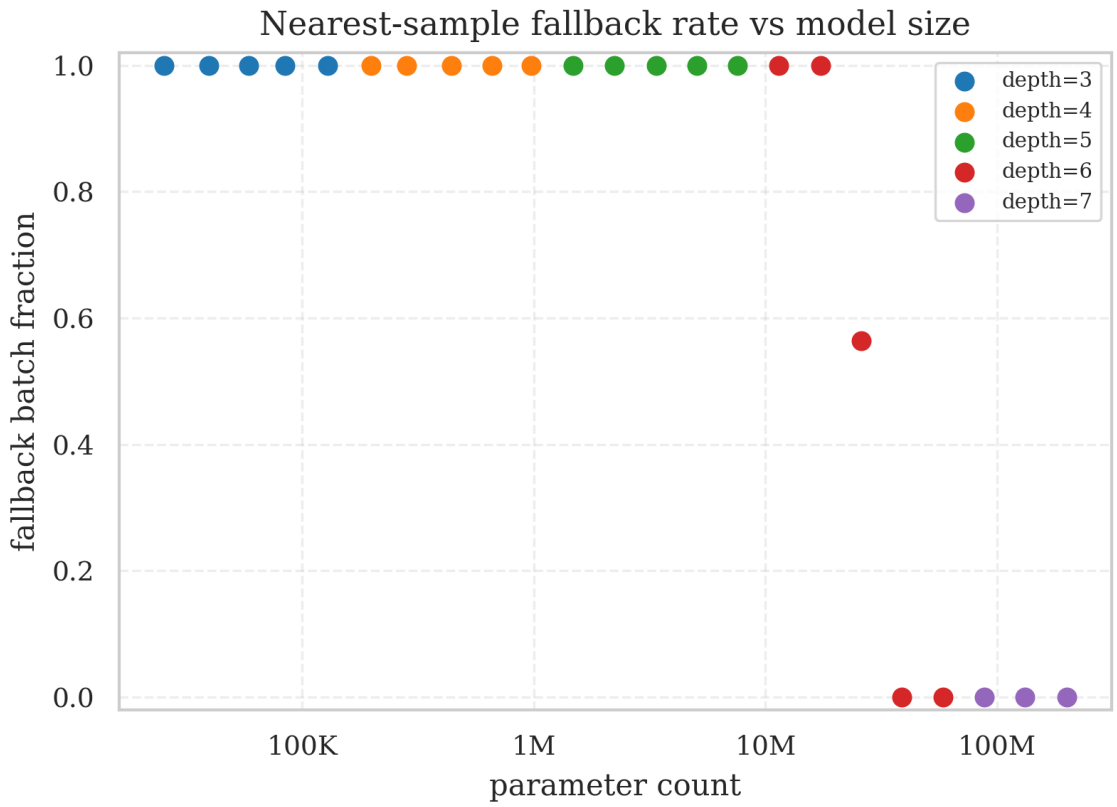

Figure 5.3: Nearest-sample fallback rate versus parameter count, colored by depth tier. The transition from 100% to 0% fallback occurs within the depth-6 tier between model-11m (34.6 ms, 100% fallback) and model-39m (119 ms, 0% fallback). Model-26m (78.8 ms) is the single transitional case at 56.3%.

Notably, the fallback boundary does not coincide simply with the 20 ms nominal polling interval. Even the depth-5 models at up to 27.8 ms and model-11m at 34.6 ms incur 100% fallback, because the OS scheduling jitter on the polling thread causes inference windows in this latency range to routinely receive zero interior samples. The transition to zero fallback occurs only once inference latency reliably exceeds the jitter range, which empirically occurs around 80–120 ms in this setup. This step function means that the EPI estimation method changes across the model family. Fast, shallow models are measured using single power samples, while slow, large models are measured using trapezoidal integration. The implications of this two-regime measurement for the interpretation of EPI values are discussed in Section 6.11.

5.5 Scaling Laws

5.5.1 EPI Scaling

Table 5.3 summarizes the measured EPI, latency, throughput, and power for all 23 benchmarked models, sorted by parameter count. EPI spans from 5,049 μJ per inference for model-25k to 4,403,000 μJ for model-200m, a factor of approximately 870×. One notable entry in the depth-6 tier is that model-39m (38.8M parameters,

0% fallback) records a lower EPI (443,800 μJ) than model-26m (25.8M parameters, 56% mixed fallback, 485,000 μJ) despite having more parameters. This inversion is attributable to the change in estimation method between these two models and is examined in Sections 6.11 and 6.4.

Table 5.3: Per-model benchmark results sorted by parameter count. EPI and its cross-repetition standard deviation (σ) are derived from the `pout1` power rail. Latency is the median over all 10,000 batches. Throughput is the reciprocal of mean batch duration. FB (fallback rate) is the fraction of batches using the nearest-sample EPI approximation. Models with FB = 0.00 use fully trapezoidal EPI integration.

Model	Params	D	EPI (μJ)	σ (μJ)	Lat. (ms)	Tput (1/s)	Pwr (W)	FB
model-25k	25K	3	5,049	23	0.813	1,211	5.53	1.00
model-39k	39K	3	5,890	36	0.909	1,053	5.77	1.00
model-59k	59K	3	6,886	26	1.075	894	5.79	1.00
model-85k	85K	3	7,991	25	1.276	760	5.77	1.00
model-129k	129K	3	9,380	45	1.488	667	5.89	1.00
model-198k	198K	4	11,100	9.5	1.721	567	6.05	1.00
model-282k	282K	4	15,000	11	2.179	450	6.53	1.00
model-440k	440K	4	20,800	61	2.839	347	7.03	1.00
model-658k	658K	4	26,280	112	3.746	264	6.80	1.00
model-973k	973K	4	36,940	186	5.187	191	6.95	1.00
model-1_5m	1.5M	5	48,110	48	6.379	156	7.40	1.00
model-2_2m	2.2M	5	70,550	1,008	9.163	108	7.56	1.00
model-3_4m	3.4M	5	104,300	233	13.03	76.5	7.93	1.00
model-5_1m	5.1M	5	148,900	130	18.89	52.8	7.83	1.00
model-7_6m	7.6M	5	224,500	78	27.76	36.0	8.05	1.00
model-11m	11M	6	273,500	51	34.57	28.9	7.88	1.00
model-17m	17M	6	423,200	176	52.01	19.2	8.11	1.00
model-26m	26M	6	485,000	1,936	78.75	12.7	8.27	0.56
model-39m	39M	6	443,800	732	118.9	8.41	8.18	0.00
model-58m	58M	6	961,300	764	183.9	5.44	8.07	0.00
model-88m	88M	7	1,472,000	683	244.2	4.10	8.20	0.00
model-132m	132M	7	2,566,000	18,950	384.2	2.60	8.04	0.00
model-200m	200M	7	4,403,000	14,760	626.5	1.60	7.84	0.00

Figure 5.4 shows EPI against parameter count on a log-log scale. The relationship is well described by the power law

$$\log_{10} E = 0.749 \cdot \log_{10} P + 0.176, \qquad R^2 = 0.984, \tag{5.1}$$

where E is EPI in μJ and P is parameter count. The exponent of 0.749 is well below 1.0, meaning that doubling the parameter count increases EPI by a factor of $2^{0.749} \approx 1.68$ rather than 2. The coefficient of determination $R^2 = 0.984$ measures how much of the total variation in EPI across models is explained by the fitted curve, on a scale from 0 (no better than predicting the mean) to 1 (perfect fit). A value of 0.984 indicates that the fitted curve explains 98.4% of observed cross-model variation in EPI, with the remaining 1.6% attributable to depth-tier banding and measurement noise. Depth-tier banding is visible around the regression line, with depth-4 models (orange) tending to sit slightly below the line, while depth-7 models (purple) sit above it, reflecting the secondary contribution of depth to EPI beyond what parameter count captures. This non-monotonic pattern is discussed further in Section 6.1.

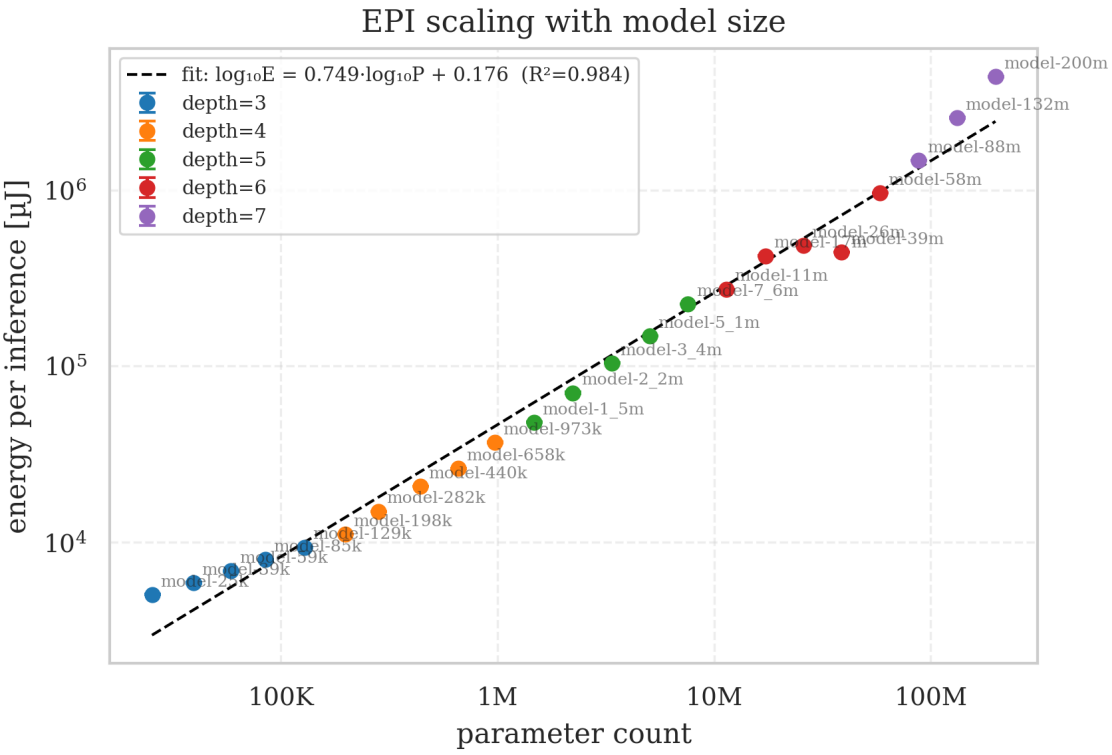

Figure 5.4: EPI (from the `pout1` rail) versus parameter count on a log-log scale, colored by depth tier. The dashed line is the fitted power law (Equation 5.1). Depth-4 models (orange) sit slightly below the line. Depth-7 models (purple) sit above it.

5.5.2 Latency Scaling

Figure 5.5 shows median inference latency against parameter count on a log-log scale. A power-law regression yields

$$\log_{10} T = 0.755 \cdot \log_{10} P - 3.689, \qquad R^2 = 0.981, \tag{5.2}$$

where T is latency in milliseconds and P is parameter count. Latency therefore scales sublinearly with parameter count across the SimNet family. This reflects that parameter count is not identical to executed work, since progressive pooling reduces the sequence length through the network, so later parameters contribute fewer MACs per inference than early parameters. Latency spans 0.813 ms (`model-25k`) to 627 ms (`model-200m`), a factor of 771×. All 23 models fall well above the B4096 compute-bound reference line at 614 GMACs/s. The measured latency exceeds the compute-bound prediction by a factor of approximately 4.6× to 32× across the family, indicating that the DPU does not reach its theoretical compute ceiling under batch-size-1 inference. The physical reason for this gap is examined in Section 6.2.

The three depth-7 models sit as a group above the global regression line, with positive latency residuals of 21%, 40%, and 67% for model-88m, model-132m, and model-200m respectively. This is a tier-level pattern. All three depth-7 models are affected, and the magnitude grows with model size within the tier.

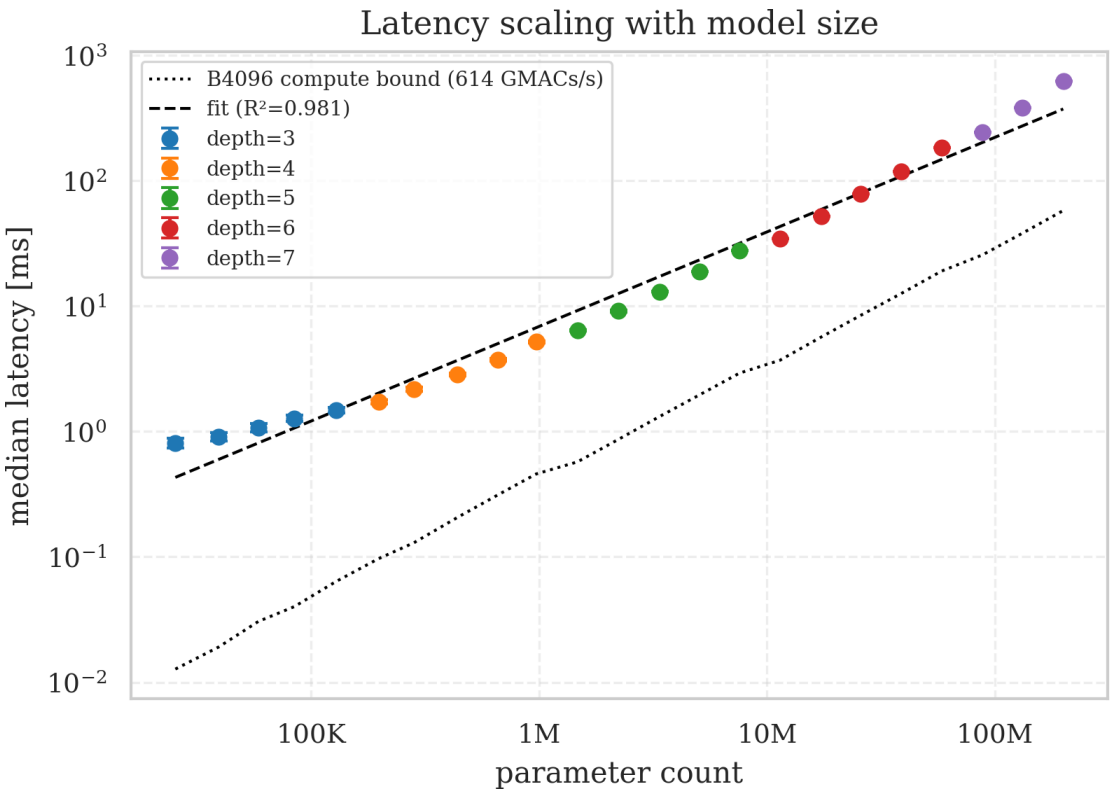

Figure 5.5: Median inference latency versus parameter count on a log-log scale, colored by depth tier. Error bars show the cross-repetition standard deviation. The dashed line is the fitted power law ($R^2 = 0.981$). The depth-7 tier (purple) sits above the fitted line as a group. The B4096 compute-bound reference is discussed in the roofline analysis (Section 5.6).

5.5.3 Throughput and Real-Time Feasibility

Figure 5.6 shows throughput (inferences per second) as a function of parameter count. Throughput is the reciprocal of mean batch latency and therefore mirrors the latency curve on an inverted log scale. The target rubber bushing application streams sensor data at 1000 Hz with a 256-sample window stride, producing one new inference window every 256 ms. A sustained throughput of $1000/256 \approx 3.9$ inferences/s would therefore suffice for the windowing rate alone. The 50 Hz figure is a system-level requirement provided by the industrial partner that exceeds the 3.9 Hz windowing rate. Note that end-to-end alert latency is dominated by the 256 ms stride rather than the inference time, so the practical benefit of exceeding the windowing rate is marginal for latency alone. The 50 Hz threshold is treated as a given deployment requirement and is not derived from the windowing setup. All 14 models up to and including `model-5_1m` (5.1M parameters, 52.8 inf/s) exceed the 50 Hz reference rate for the target rubber bushing sensor application. `model-7_6m` at 36.0 inf/s falls below this threshold. The five depth-3 models achieve between 667 and 1,211 inferences/s. The two largest models, `model-132m` and `model-200m`, achieve 2.60 and 1.60 inferences/s respectively and cannot support real-time streaming at 50 Hz.

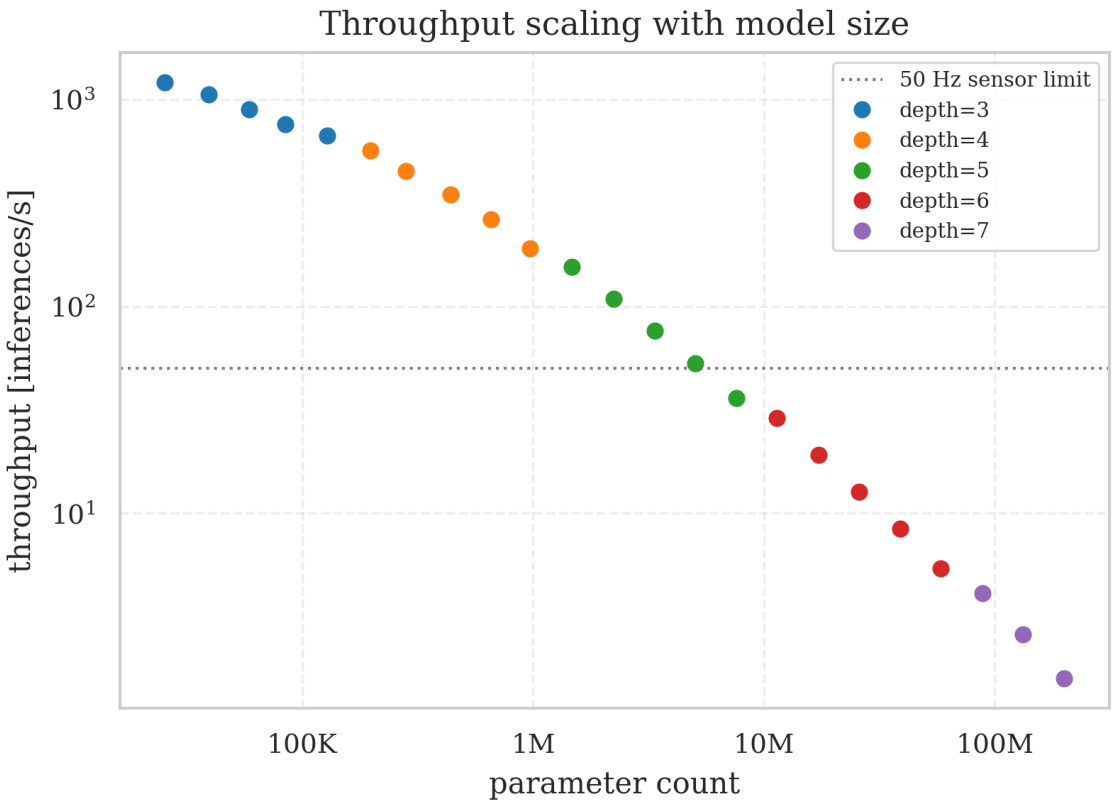

Figure 5.6: Throughput (inferences per second) versus parameter count on a log-log scale, colored by depth tier. The dotted horizontal line marks the 50 Hz real-time threshold.

5.5.4 Power Scaling

While EPI and latency follow tight power laws, power shows a qualitatively different pattern. Figure 5.7 shows mean `pout1` power during inference rising from approximately 5.53–5.89 W for depth-3 models to a plateau of approximately 7.8–8.3 W for depth-6 and depth-7 models, with a family mean of 7.19 W. Beyond this plateau, increasing model size continues to raise EPI through increasing latency, but not through increasing power. The sublinear EPI exponent mainly follows from sublinear latency scaling, while the power plateau prevents model size from adding a further power-growth component. The depth-3 tier shows anomalously low power relative to depth-4 models even at comparable parameter counts. Two competing explanations for this anomaly are evaluated in Section 6.3. Within the depth-6 tier, model-11m (11.4M parameters, 7.88 W) sits noticeably below its neighbors model-17m (8.11 W) and model-26m (8.27 W).

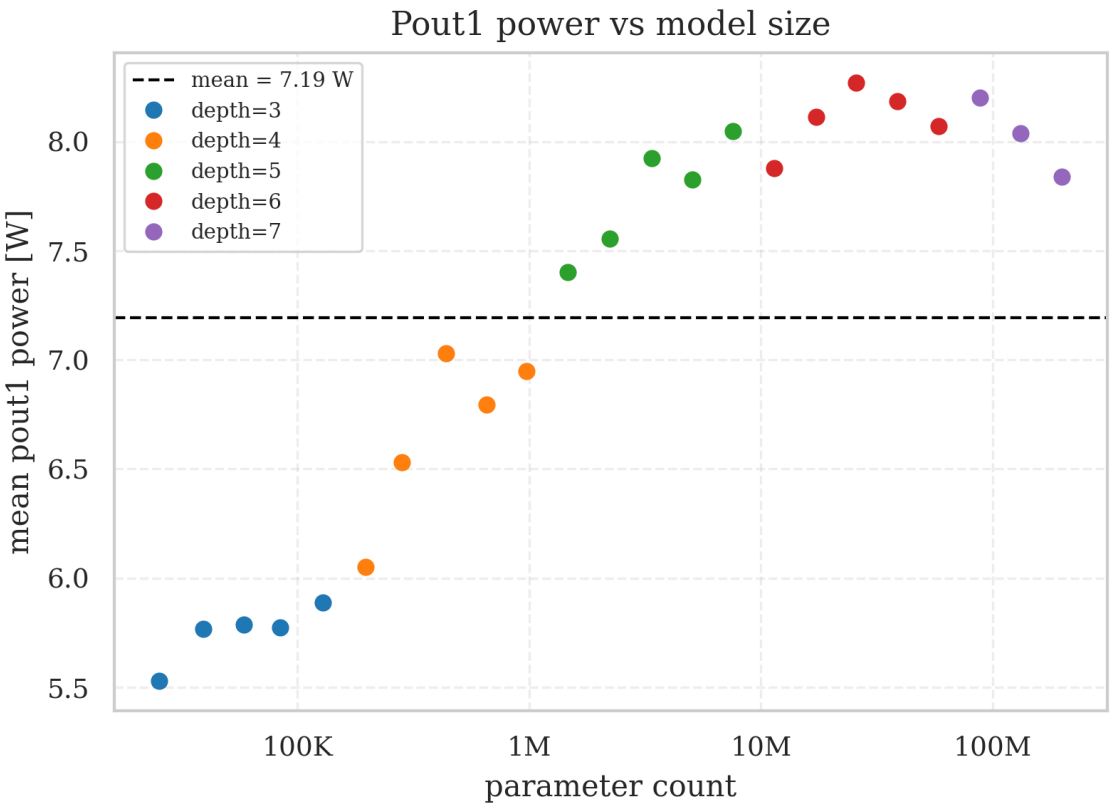

Figure 5.7: Mean `pout1` power during inference versus parameter count, colored by depth tier. The dashed line marks the family mean (7.19 W). Power saturates around 8.1–8.3 W for depth-6 and depth-7.

5.6 Roofline Analysis

The latency results in Section 5.5.2 show that the measured models run much slower than the theoretical B4096 compute peak would suggest. The roofline analysis is used as a diagnostic check to identify whether this gap is caused by memory bandwidth or by poor utilization of the DPU compute units.

The roofline plot compares two quantities. The horizontal axis shows arithmetic intensity, measured as MACs per byte of model and activation data. This indicates how much computation is performed for each byte moved through the system. The vertical axis shows achieved compute throughput, measured as total MACs divided by measured inference latency. The B4096 has a theoretical compute ceiling of 614 GMACs/s and a memory bandwidth ceiling of 19.2 GB/s. These two ceilings meet at 32 MACs/byte, known as the ridge point.

The interpretation is as follows. Models to the left of the ridge point are memory-bound, meaning that memory movement limits performance. Models to the right of the ridge point are compute-bound, meaning that memory bandwidth is not the main limiting factor and the model should instead be limited by how effectively the DPU compute units are used.

All 23 SimNet models fall to the right of the ridge point, with arithmetic intensities

between approximately 80 and 320 MACs/byte. They are therefore compute-bound according to the roofline model. However, their achieved throughput is still far below the 614 GMACs/s B4096 peak, ranging from approximately 19 to 133 GMACs/s. This means that the models are not slow because they lack memory bandwidth. They are slow because the DPU compute units are not fully utilized in this operating mode.

This result supports the interpretation used in Section 6.2. SimNet is a 1D CNN with short feature maps that become even shorter after repeated MaxPool1d operations. This gives the DPU too little spatial work to keep its tiling pipeline fully occupied, especially at batch size one. The roofline result therefore shows that peak GMACs/s is a poor proxy for the actual throughput of this 1D time-series workload on the B4096.

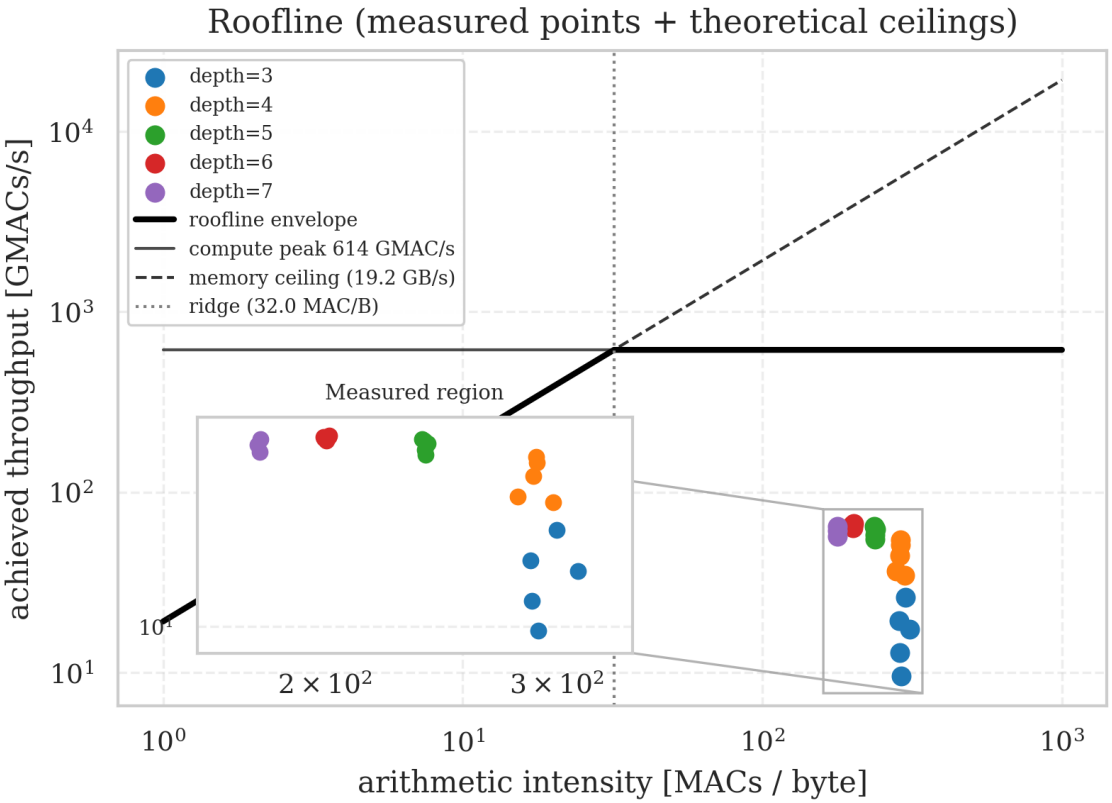

Figure 5.8: Roofline plot for the 23 benchmarked models on the B4096. The ridge point at 32 MACs/byte separates memory-bound models from compute-bound models. All measured models are to the right of this point, so memory bandwidth is not the main bottleneck. Even so, they remain far below the 614 GMACs/s compute ceiling, showing that the DPU compute units are not fully utilized for this batch-size-1 1D CNN workload.

5.7 Feature Analysis

The scaling laws in Section 5.5 establish what happens across the model family. This section establishes *why*, by identifying which architecture features are most strongly associated with the observed performance targets. This analysis also motivates the

feature engineering choices for the predictive model.

Figures 5.9 and 5.10 show Pearson and Spearman correlations between a representative set of architecture features and the three prediction targets. Log-transformed parameter count achieves a Spearman ρ of 1.00 with latency and 0.999 with EPI (rounding to 1.00 in the figure), and $\rho = -1.00$ with throughput, reflecting a perfect monotone rank relationship with all three targets across the 23 models. Depth achieves $\rho = 0.979$ with EPI. The scatter matrix (Figure 5.11) makes the near-perfect log-log linearity between `log10(MACs)` and EPI ($R^2 = 0.98$) and between `log10(base_width)` and EPI ($R^2 = 0.98$) visually apparent, while depth alone explains less variance ($R^2 = 0.95$).

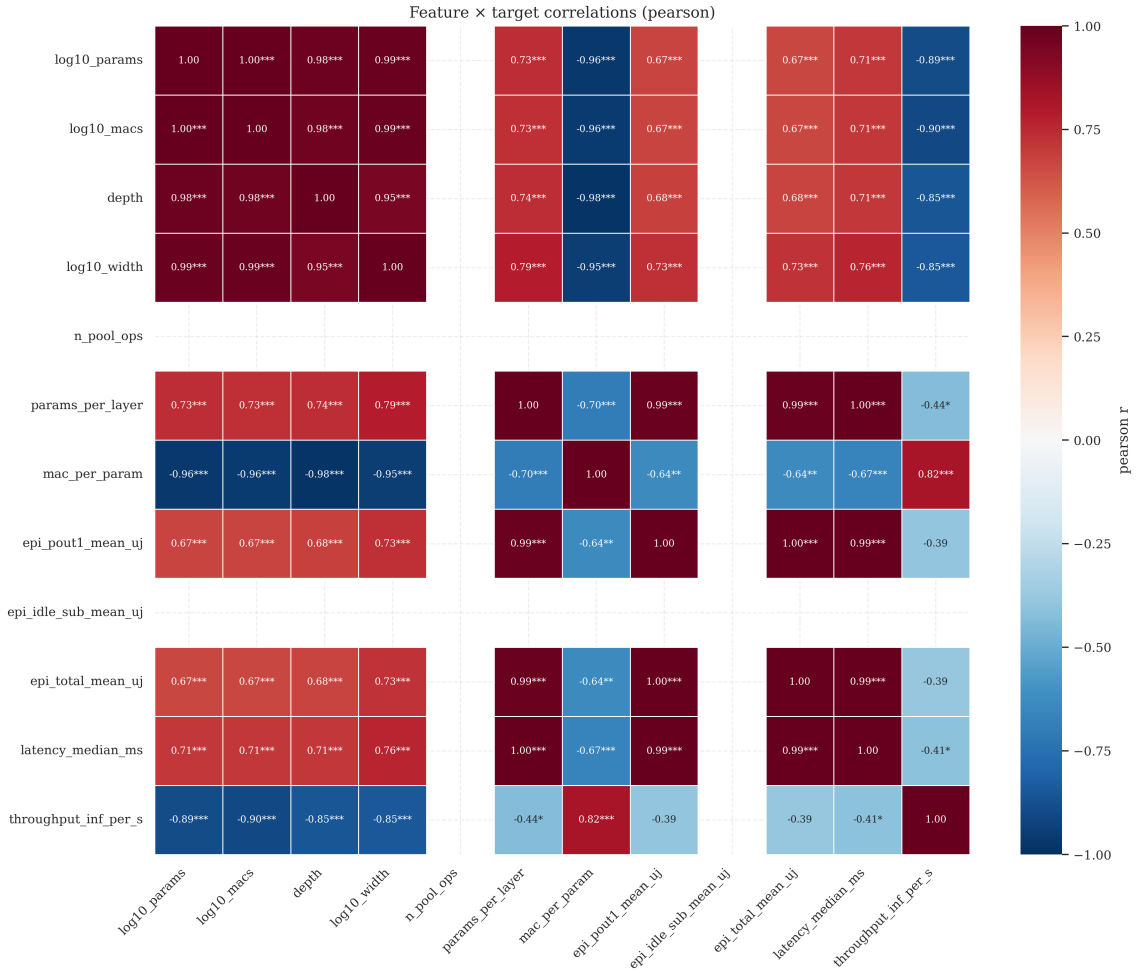

Figure 5.9: Pearson correlation matrix between selected architecture features and prediction targets. Stars denote statistical significance: $^{***}p < 0.001$, $^{**}p < 0.01$, $^{*}p < 0.05$.

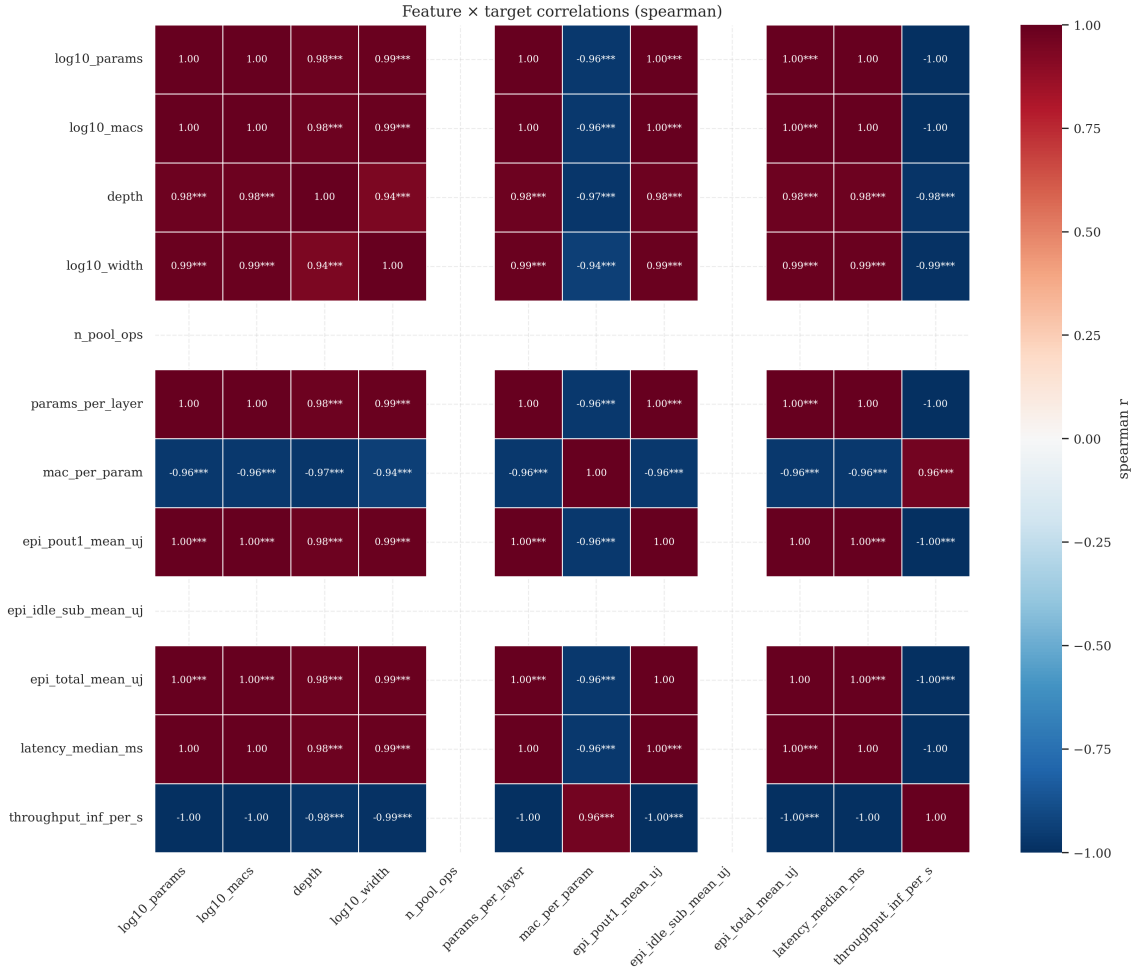

Figure 5.10: Spearman rank correlation matrix. The near-unity magnitudes for size-related features reflect the near-perfect monotone ordering of EPI, latency, and throughput with parameter count.

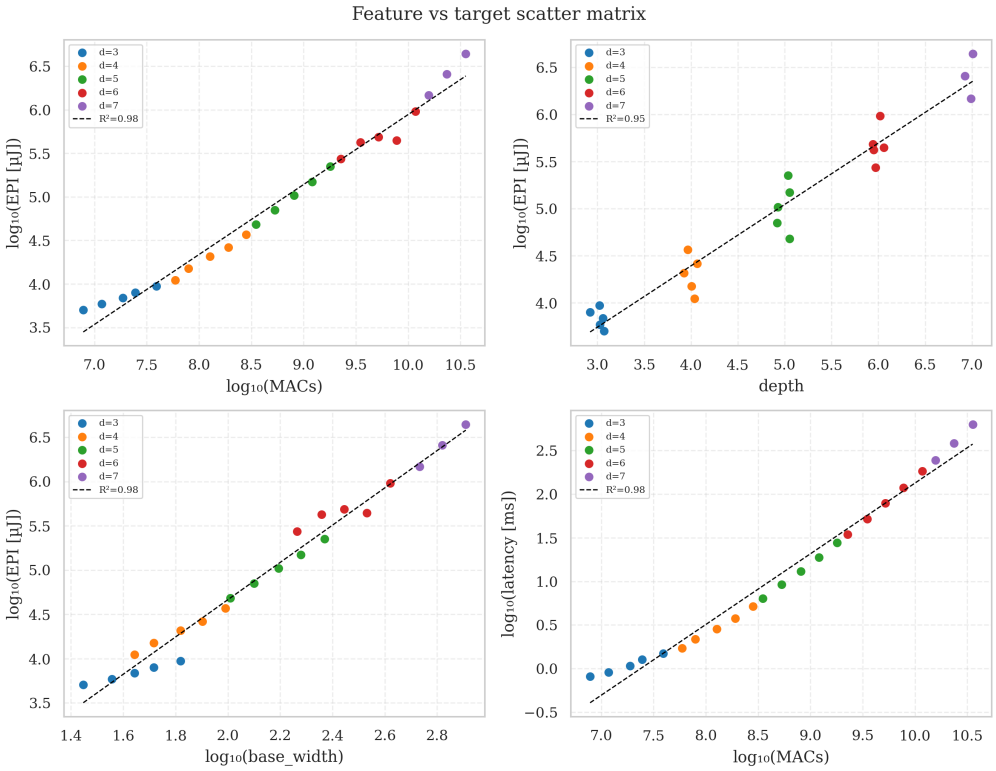

Figure 5.11: Scatter matrix of log(EPI) and log(latency) against key predictors, colored by depth tier. Dashed lines show ordinary least-squares fits with R^2 values indicating how well each single feature explains the target.

The feature clustering dendrogram (Figure 5.12) shows the result of hierarchical clustering on pairwise Spearman distances ($1 - |\rho|$) at a cut-distance of 0.15 (equivalent to $|\rho| \geq 0.85$). The 29 candidate features collapse into four clusters. Cluster 1 contains 25 features, including all size-and-compute features (`log10_params`, `log10_macs`, `log10_width`, `depth`, `param_count`, and their derivatives), mutually correlated at $|\rho| \geq 0.94$. This extreme collinearity means that any one feature from this cluster carries almost identical information for prediction purposes. Cluster 2 contains pooling and kernel-geometry features ($|\rho| = 0.86$ with EPI). Cluster 3 is a singleton (`mean_kernel_size`, $\rho = 0.12$ with EPI). Cluster 4 is a singleton (`arithmetic_intensity`, $\rho = -0.09$ with EPI). The four clusters provide four independent candidate features for the predictive model's forward selection stage.

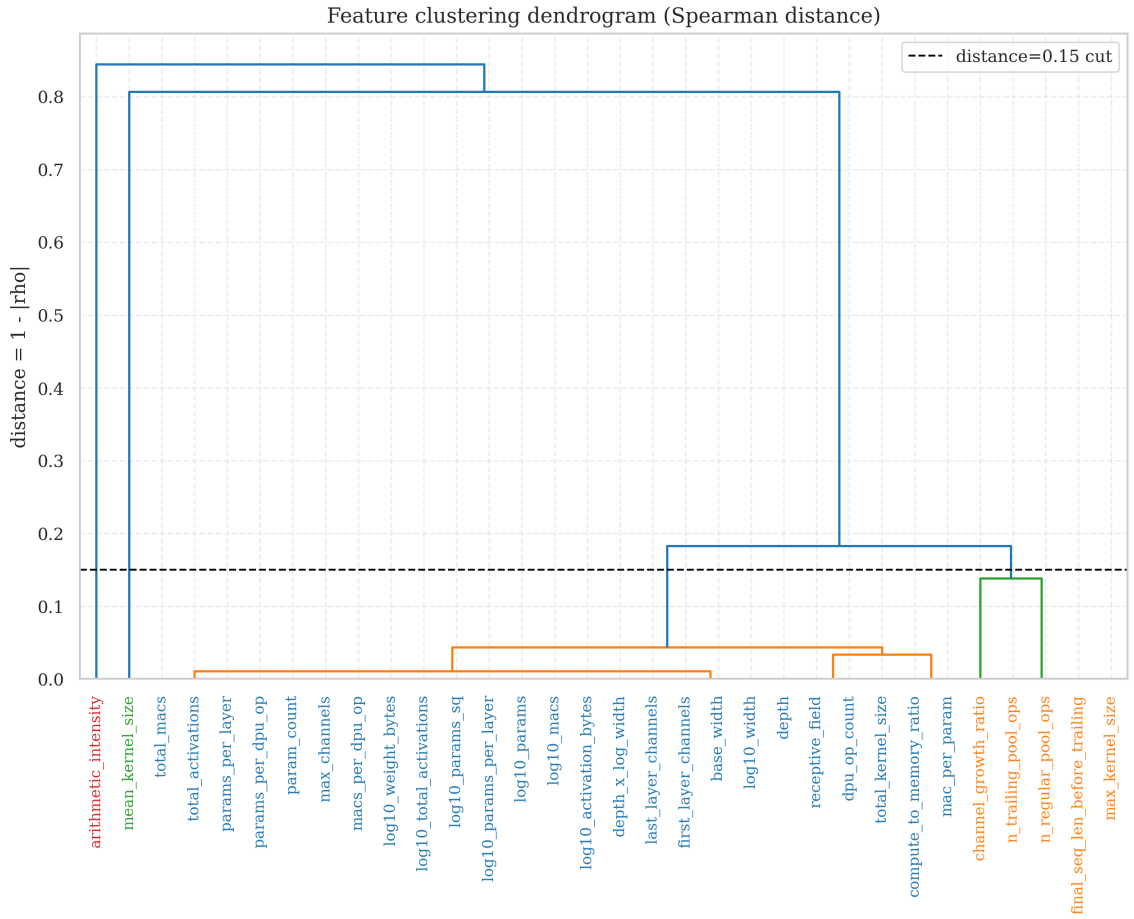

Figure 5.12: Hierarchical clustering dendrogram on pairwise Spearman distances for all 29 candidate features, cut at distance 0.15. Features are colored by cluster. Cluster 1 (blue, 25 features) encompasses all size-and-compute features. Cluster 2 (orange) covers pooling and kernel-geometry features. Clusters 3 and 4 are singletons.

5.8 Predictive Modeling

5.8.1 Feature Selection

Forward feature selection evaluated one cluster representative per cluster for each target under leave-one-out cross-validation (LOO-CV), using mean absolute percentage error (MAPE) as the selection criterion. Figure 5.13 shows the resulting paths.

For EPI, the selector chose the depth-width interaction term `depth_x_log_width` ($= D \times \log_{10} w$, the product of depth and the log of base channel width) as the single selected feature, achieving LOO-CV MAPE of 24.9% and $R^2 = 0.792$. No second candidate reduced MAPE, so selection halted at step 1. For latency, `log10_params` was selected as the sole feature with LOO-CV MAPE of 26.0% and $R^2 = 0.974$. Again, no second candidate helped. For throughput, the selector first chose `depth_x_log_width` (MAPE 105.8%), then added `mean_kernel_size` as a second feature, bringing MAPE to 33.1% and R^2 to 0.995. The fact that EPI requires an interaction term while latency is fully determined by a single size measure reflects the two-component nature

of EPI, and is interpreted in Section 6.1.

Fig. 15: Forward feature selection diagnostics

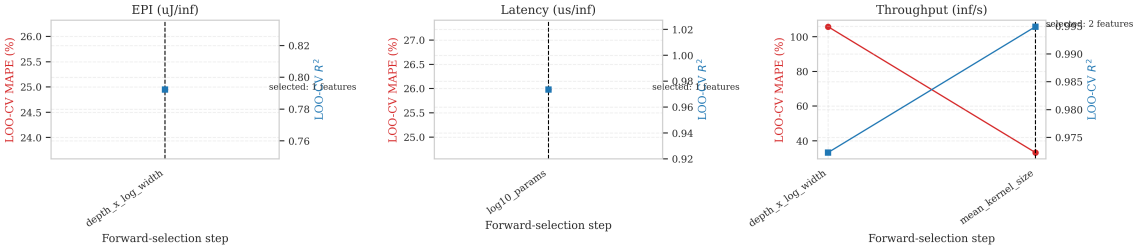

Figure 5.13: Forward selection paths for EPI (left), latency (center), and throughput (right). Red lines show LOO-CV MAPE (left axis). Blue lines show R^2 (right axis). Vertical dashed lines mark the stopping point. EPI and latency each require only one feature. Throughput requires two.

5.8.2 Model Comparison

Figure 5.14 compares five candidate estimators under LOO-CV for all three targets using MAPE. The naive mean baseline achieves MAPEs of 2440%, 2472%, and 2422% for EPI, latency, and throughput respectively, representing the minimum bar any model must clear. Linear and Ridge regression reduce MAPE substantially to around 1841% (EPI), 1998% (latency), and 974% (throughput), but remain in the same order of magnitude as the naive baseline. Degree-2 polynomial Ridge achieves 837%, 1400%, and 270%. Gaussian process regression (GPR) with an anisotropic automatic relevance determination radial basis function (ARD-RBF) kernel achieves MAPEs of 24.9%, 26.0%, and 33.1%, a reduction of 98% or more relative to the naive baseline for all three targets and a factor of 35 or more relative to the next-best estimator (polynomial Ridge). All reported MAPEs are lower bounds because the feature selection procedure is partially nested.

Fig. 13a: Overall model comparison (LOO-CV)

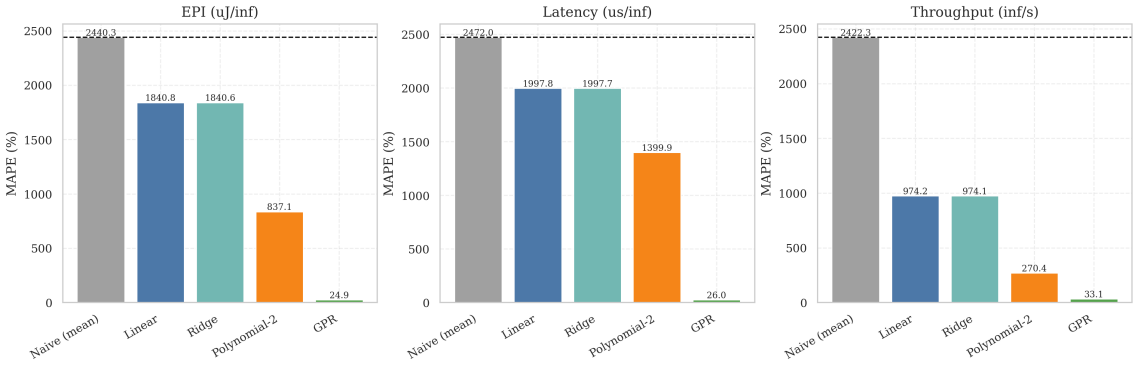

Figure 5.14: LOO-CV MAPE for five estimator types across EPI, latency, and throughput. The dashed horizontal line marks the naive mean baseline. GPR outperforms all alternatives by one to two orders of magnitude relative to the naive baseline, and by a factor of 35 or more relative to polynomial Ridge.

5.8.3 GPR Prediction Accuracy

The detailed LOO-CV results for the GPR are summarized in Table 5.4. Four complementary views of these results are provided in Figures 5.15 through 5.18.

Table 5.4: GPR LOO-CV prediction metrics. All MAPEs are stated as lower bounds due to partial nesting of feature selection (Section 4.7). The throughput log-MAPE (93.3%) is dominated by a small-denominator artifact. Depth-7 models achieve only 1.6–4.1 inf/s, and log-MAPE's percentage-error calculation amplifies small absolute residuals when the actual value is this low. For example, a prediction error of 0.5 inf/s at an actual throughput of 2 inf/s produces a log-MAPE contribution of 25%, whereas the same absolute error at 100 inf/s contributes only 0.5%. MdAPE and R^2 are the recommended metrics for throughput.

Target	MAPE (%)	log-MAPE (%)	MdAPE (%)	R^2	ρ_S
EPI (μJ/inf)	24.9	2.09	13.8	0.792	0.977
Latency (ms/inf)	26.0	2.59	12.6	0.974	0.963
Throughput (inf/s)	33.1	93.3	15.4	0.995	0.976

Figure 5.15 shows predicted versus actual values on the identity diagonal for all three targets. For EPI, most models cluster tightly around the diagonal, with the depth-6 tier (red) showing the most spread, consistent with the within-tier heterogeneity described in Section 5.5.4. The depth-7 models (purple) appear consistently above the diagonal in both EPI and latency panels, confirming the tier-level positive bias noted in Section 5.5.2. The throughput panel shows very tight alignment along the diagonal for all but the slowest (largest) models, consistent with its $R^2 = 0.995$.

Fig. 10: Predicted vs Actual (LOO-CV)

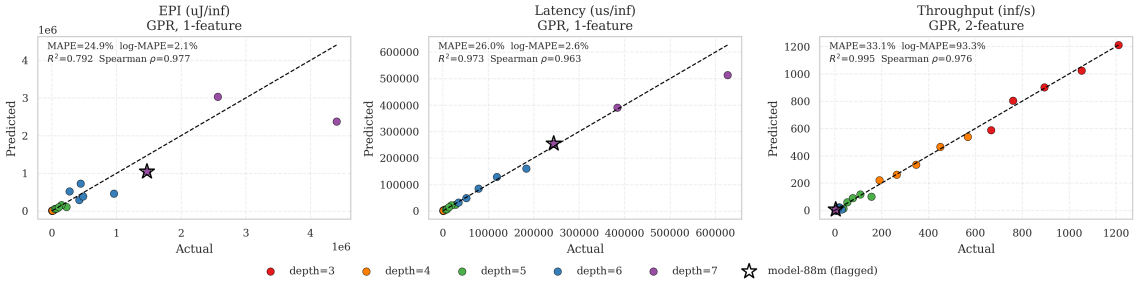

Figure 5.15: LOO-CV predicted versus actual values for EPI (left), latency (center), and throughput (right), colored by depth tier. Points above the dashed identity line are over-predicted. Points below are under-predicted. The depth-7 tier (purple) shows a consistent upward bias in EPI and latency.

Figure 5.16 shows the residuals (actual minus predicted) plotted against predicted values, overlaid with a locally weighted scatterplot smoothing (LOWESS) trend line. For EPI, the trend is approximately flat across most of the predicted range, with an upward deviation at the high end corresponding to the depth-7 models. For latency, a similar upward deviation at the high end confirms the tier-level bias. For throughput,

the residuals are scattered without systematic trend, consistent with the near-zero LOWESS line across the full predicted range.

Fig. 11: Residuals with LOWESS trend (LOO-CV)

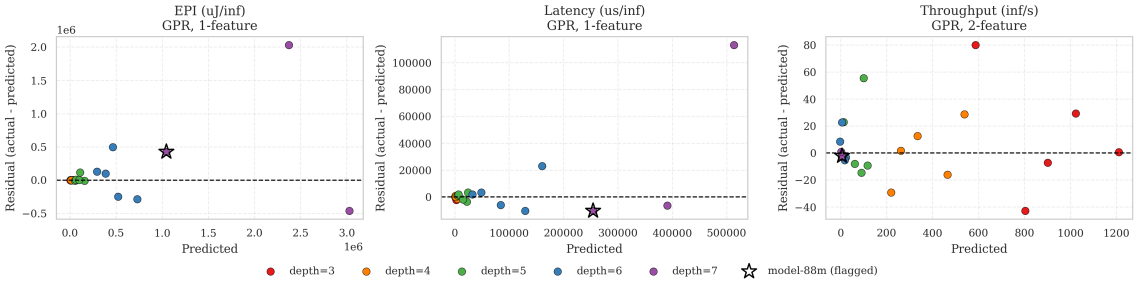

Figure 5.16: LOO-CV residuals (actual minus predicted) versus predicted value for EPI (left), latency (center), and throughput (right), with LOWESS trend lines. The upward deviation at the high end of the EPI and latency panels corresponds to the depth-7 tier.

Figure 5.17 provides a comprehensive view of all five estimators across all five metrics and three targets. The heatmap shows a clear separation. The GPR row is uniformly green (low error) while all other estimators are predominantly red for MAPE. Notably, Spearman rank correlation (rightmost column) is high for several estimators, indicating that even simpler models can rank architectures in approximately the right order, but only the GPR achieves both correct ranking and low absolute error.

Fig. 16: LOO-CV metric summary

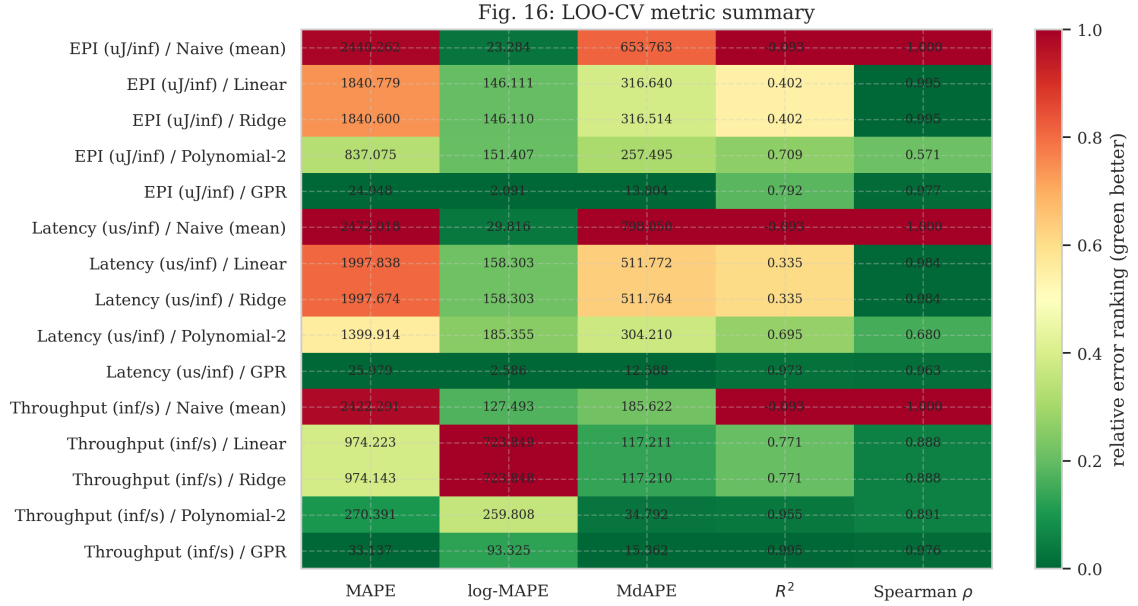

Figure 5.17: LOO-CV metric heatmap for all estimator-target combinations. Green indicates better performance. Red indicates worse. The GPR row is uniformly the best across all targets and metrics for absolute error, while Spearman rank correlation is more competitive across estimators.

Figure 5.18 shows the 95% prediction intervals (PIs) for EPI, with models sorted by actual EPI. The intervals are narrow for the smaller models (models 1–15 in the sorted order, corresponding to depth-3 through depth-5) and widen progressively for the larger, sparser models in the depth-6 and depth-7 range. This widening reflects the GPR's expected model-based assessment that predictions become less certain as models grow beyond the densely sampled region of the parameter space. Twenty-two of the 23 actual values fall within their respective 95% PI, giving an empirical coverage of 95.7%, consistent with the nominal 95% level at $N = 23$.

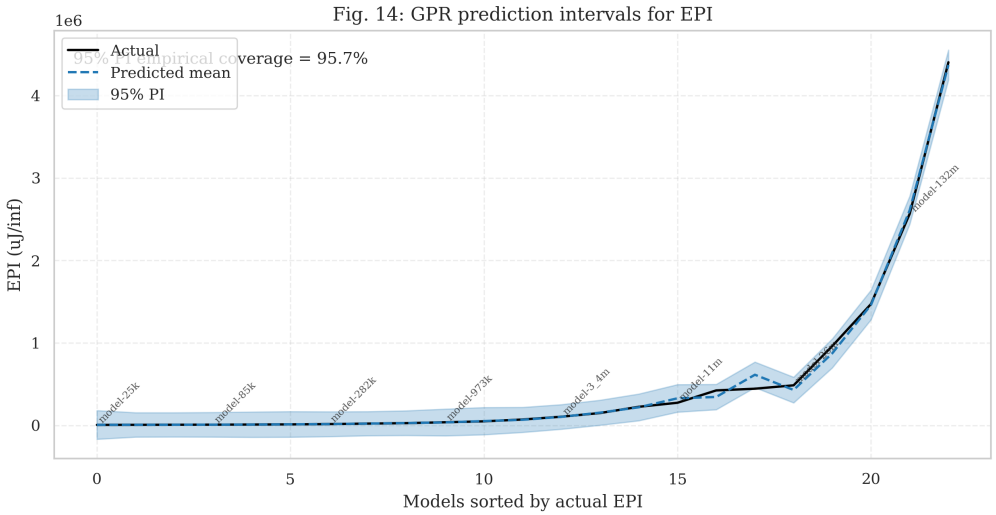

Figure 5.18: GPR EPI predictions with 95% prediction intervals, models sorted by actual EPI. The solid line is the measured value. The dashed line is the GPR posterior mean, and the shaded band is the 95% PI. Intervals widen for larger models in the sparse depth-6 and depth-7 range. Empirical coverage is 95.7%.

5.8.4 Depth-Stratified Evaluation

Figure 5.19 shows GPR LOO-CV MAPE broken down by depth tier, revealing the global model's interpolation accuracy within individual tiers. For depth-3, EPI MAPE is approximately 2% and latency and throughput MAPEs are near zero. The GPR interpolates within this tier with near-perfect accuracy. Error increases monotonically through depth-4 (13–15% across targets) and depth-5 (15–16%). The depth-6 tier shows high EPI MAPE (approximately 54%) driven by the within-tier heterogeneity introduced by model-26m's fallback transition and model-11m's anomalous power, while latency and throughput MAPEs remain more modest (18% and 12%). The depth-7 tier shows the highest errors (EPI 68%, latency 57%, throughput 56%), but contains only three models. LOO-CV within this tier trains on two points and evaluates on one, so these figures describe interpolation between two models rather than genuine generalization. They are reported for completeness and should not be interpreted as prediction accuracy estimates.

Fig. 13b: Depth-stratified error (GPR, LOO-CV)

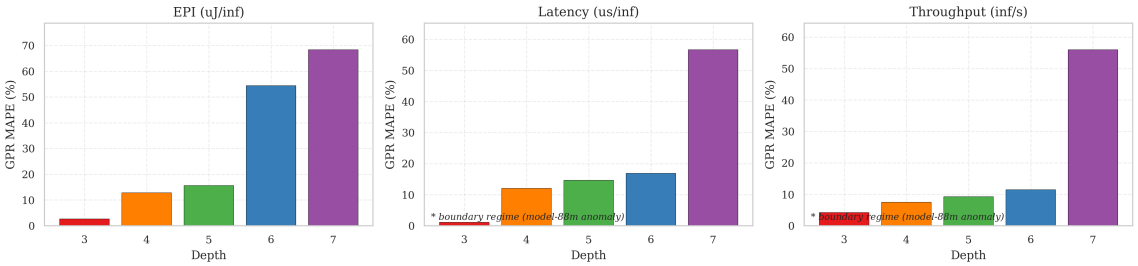

Figure 5.19: GPR LOO-CV MAPE by depth tier for EPI (left), latency (center), and throughput (right). Within-tier error increases with depth, with the depth-7 results ($N = 3$, LOO trains on 2 points) indicating only interpolation performance, not generalization.

5.8.5 EPI Regime Analysis

The anomalous power behavior of depth-3 models motivated an exploratory analysis of whether treating depth-3 as a separate prediction regime improves accuracy for the remaining models. This analysis was conducted after observing the depth-3 power anomaly and is explicitly exploratory. It was not pre-registered.

Figure 5.20 shows residuals from the full 23-model GPR, split by depth-3 (red) and depth≥4 (blue). The full model produces MAPE of 19.7% within the depth-3 group and 26.4% within the depth≥4 group. Training a separate GPR on the 18 depth≥4 models with a two-feature specification (`depth_x_log_width` as the primary predictor and `arithmetic_intensity` as an exploratory second feature. No principled basis for a second feature exists once `depth_x_log_width` is selected, and `arithmetic_intensity` was chosen as the available singleton with the most distinct roofline interpretation) yields MAPE 31.7% on that subset, worse than the 26.4% from the full model. The regime split does not improve prediction accuracy, and the single full-model GPR remains the recommended predictor. The analysis confirms, however, that depth-3 constitutes a qualitatively distinct operating regime in residual space, consistent with the power and fallback-rate observations.

Fig. 17: EPI regime analysis

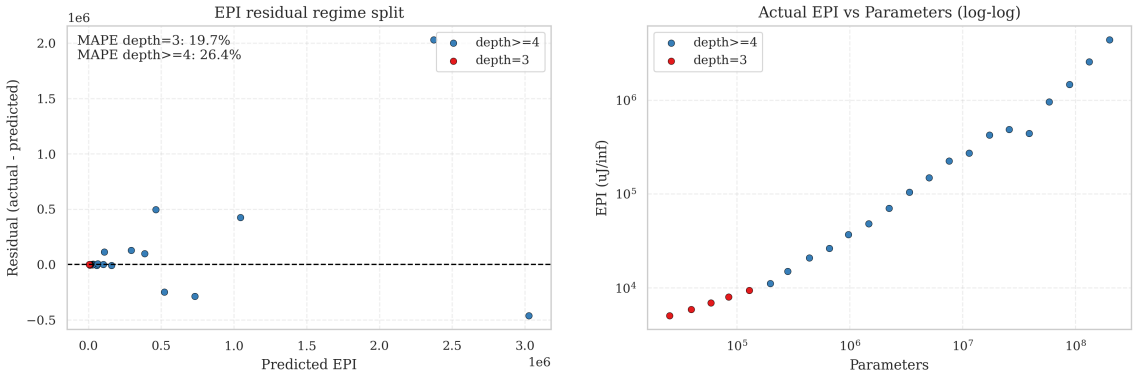

Figure 5.20: EPI regime analysis. Left panel shows residuals from the full 23-model GPR, colored by depth= 3 (red) and depth≥ 4 (blue), with within-group MAPEs annotated. The right panel shows log-log EPI versus parameter count using the same color scheme. The depth-3 group (red) sits systematically below the trend for deeper architectures.

Chapter 6

Discussion

This chapter interprets the findings from Chapter 5 in the context of the research questions and the broader literature. It is structured into ten sections: (1) answering the research questions, assembling the evidence into direct answers with physical reasoning; (2) why the DPU does not reach its theoretical peak, explaining the utilization gap and its consequences for the scaling exponent; (3) measurement validity, examining the nearest-sample estimator and the choice of the `pout1` metric; (4) the depth-3 power anomaly, evaluating two competing explanations; (5) depth-tier heterogeneity and the model-39m EPI inversion; (6) why GPR outperforms simpler models at $N = 23$; (7) feature interpretability and what the selected features reveal about how the DPU encodes architecture; (8) the compiler as a hidden deployment risk; (9) connections to the identified research gaps; and (10) generalization, limitations, and future work. The chapter closes with a discussion of the ethical, societal, and sustainability dimensions of the work.

6.1 Research Questions Revisited

RQ1: How can EPI, latency, and throughput be estimated from model configuration parameters before hardware benchmarking?

RQ1 is answered by the three GPR predictors, each trained on a single physically motivated feature selected by forward LOO-CV from a de-collinearized set of 29 architecture features grouped into four clusters. For EPI, the selected feature is the engineered interaction term `depth_x_log_width` ($= D \times \log_{10} w$). This term outperformed `log10_params` on LOO-CV MAPE despite the latter achieving higher Spearman ρ with EPI. `log10_params` captures total compute load but conflates depth and channel width across depth tiers, while the interaction term encodes the joint effect of depth and channel scale on measured power draw, which is the component of EPI that `log10_params` alone cannot adequately represent. For latency, `log10_params` is the selected feature, reflecting that latency is dominated by compute load and a single size measure suffices. For throughput, a two-feature model using `depth_x_log_width` and `mean_kernel_size` is selected, since kernel size determines per-layer DPU cycle count and provides marginal additional signal. Deployability under the Vitis AI compiler must be verified before any EPI prediction is meaningful. Architectures that fail the pre-flight DPU-subgraph check fall outside the predictor's domain of validity.

RQ2: What relationships exist between CNN architecture parameters and observed EPI on the ZCU104 platform?

The answer is an empirical power-law scaling relationship ($E \propto P^{0.749}$, $R^2 = 0.984$) across four orders of magnitude. The sublinear exponent is not arbitrary. It follows from two measured behaviors in the platform. First, latency itself scales sublinearly with parameter count because progressive pooling reduces the sequence length through the network, so parameter count grows faster than executed work. Second, power does not scale linearly with parameter count. It rises from approximately 5.5 W at depth-3 to a plateau of approximately 8.2 W at depth-6 and depth-7. The power plateau prevents larger models from adding a separate power-growth exponent, so EPI mostly tracks the latency exponent. This shape is the primary answer to RQ2.

Depth carries independent information beyond parameter count. The feature selection result captures this asymmetry precisely. Latency is almost completely determined by `log10_params` alone ($R^2 = 0.974$ with a single feature), because MAC execution time depends only on compute load. EPI required the depth-width interaction term `depth_x_log_width` instead, because power depends not only on how much computation is performed, but also on how many DPU pipeline stages are engaged per inference. A deeper model at the same parameter count draws slightly more power because more instruction stages are exercised, and the interaction term $D \times \log_{10} w$ encodes this joint dependency efficiently. The asymmetry between latency and EPI predictability is therefore not a modeling artifact. It is a direct consequence of EPI's two-component nature.

The depth-tier banding in the EPI scaling figure is non-monotonic and deserves a careful reading. Depth-4 models sit below the global regression line (mean residual -19%), while depth-7 models sit above it (mean residual $+44\%$). This is not a simple rule that deeper means higher EPI at equal parameter count. Rather, it reflects the interaction of depth, channel width, and DPU instruction scheduling. Depth-4 models at 198K–973K parameters use modest channel widths with two pooling steps, which produces an efficient DPU utilization pattern, while depth-7 models at 88M–200M parameters accumulate additional overhead from seven layers of scheduling that grows with model size. Depth-3 models, paradoxically, also show positive residuals (above the line), which connects to the measurement uncertainty discussed in Section 6.3.

RQ3: How accurately can EPI, latency, and throughput be estimated?

RQ3 is answered by the LOO-CV metrics. The GPR achieves MdAPEs of 13.8%, 12.6%, and 15.4% for EPI, latency, and throughput. These figures are lower bounds due to partial nesting of feature selection. The magnitude of this inflation is unknown, although the small number of candidate features and their high Spearman correlations with the target likely moderate the effect. The true EPI MAPE under a fully nested protocol is unknown but expected to be substantially higher than the reported lower bound, while MdAPE and log-MAPE remain more robust given their insensitivity to outliers. The elevated MAPEs (24.9% and 26.0% for EPI and latency) relative to MdAPE are driven by the depth-7 tier, which sits above the regression line as a group and contributes disproportionately to the mean absolute error. The depth-stratified evaluation (Figure 5.19) shows that within-tier latency MAPEs remain below 20% for depth-3 through depth-6.

In practical terms, the log-MAPEs of 2.09% (EPI) and 2.59% (latency) indicate that the GPR captures the dominant log-scale trends for EPI and latency. For architectures that differ by an order of magnitude in energy cost, this accuracy is adequate for preliminary screening and ranking. It is not a basis for precise energy budgeting.

The 13–15% MdAPE and the 95.7% empirical coverage of the 95% prediction intervals provide model-based uncertainty estimates for candidate architectures without hardware access, but final power allocation still requires hardware measurement.

6.2 Why the DPU Does Not Reach Its Theoretical Peak

All 23 models achieve only approximately 3–22% of the B4096 peak compute throughput despite being compute-bound by arithmetic intensity. The likely physical reason lies in the interaction between the DPU's spatial tiling architecture and SimNet's progressive pooling strategy.

The B4096 DPU is designed and optimized for workloads where the spatial dimension of feature maps remains large through many layers, allowing the tiling pipeline to stay continuously occupied. SimNet applies a `MaxPool1d(2)` operation every two convolutional layers, halving the sequence length at each step. Starting from 512 samples, the feature map is reduced to 256, 128, 64, 32, and eventually 1 over the depth of the network, followed by a trailing chain of pooling operations that collapse any remaining length to a scalar. Later convolutional layers therefore operate on sequences of 32 or fewer elements, leaving the spatial tiling hardware substantially underutilized for those layers. The dominant per-inference cost is loading weights and executing a short burst of computation per layer, not sustained high-throughput compute.

This structural property has a direct implication for the EPI scaling exponent. A DPU running at sustained 614 GMACs/s utilization would produce a different latency-versus-parameter slope, and consequently a different EPI exponent. The observed exponent of 0.749 partly reflects the short-feature-map, overhead-dominated regime specific to this architecture and operating point. Longer input sequences or less aggressive pooling would shift the operating points upward on the roofline and would likely produce both a different exponent and different absolute EPI values.

The batch-size-1 operating mode amplifies this effect. For applications that queue multiple inference requests, the per-inference dispatch overhead is amortized across the batch, DPU utilization increases, and the EPI scaling behavior may differ substantially from what is reported here. The results in this study are specific to single-inference streaming operation and should not be extrapolated to batched deployment without additional measurement.

The utilization gap also carries a practical design implication for 1D CNN architectures on DPU platforms. The short input sequence length of 512 samples is a fundamental constraint of the time-series setting. At the 1000 Hz signal acquisition rate, 512 samples span only 0.512 seconds per window, and the 50 Hz real-time constraint refers to the inference cadence (one inference per 20 ms), not the signal frequency. Increasing the sequence length would require either a higher-rate sensor or a longer analysis window, both of which change the sensing setup rather than just the model. This makes the overhead-dominated, low-utilization regime a structural feature of 1D CNN deployment on DPUs optimized for image workloads, not a consequence of an unlucky hyperparameter. Image CNNs routinely operate

on 224×224 or larger spatial grids, giving the tiling pipeline thousands of spatial positions to process per layer. A 512-sample 1D CNN gives it at most 512 positions at the first layer and 32 or fewer by the third or fourth layer. The implication is that DPU platforms designed for image inference may be structurally mismatched to 1D time-series workloads at batch size 1, and any platform comparison for embedded time-series AI should account for this possibility rather than treating peak GMACs/s as a reliable proxy for time-series throughput.

6.3 The Depth-3 Power Anomaly

The depth-3 models consume approximately 1 W less `pout1` power than the smallest depth-4 model at comparable parameter counts, producing power values of 5.53–5.89 W against 6.05 W and above for depth-4. Two explanations are plausible and not mutually exclusive.

The first is a genuine architectural effect. Depth-3 models complete each forward pass in under 1.5 ms. If the B4096 DPU pipeline requires some time to fully activate all its functional units, models that complete faster may not reach the sustained power draw of deeper models. This would make the depth-3 power anomaly a real feature of how shallow models engage the DPU, consistent with the roofline finding that all models operate well below the 614 GMACs/s ceiling. Even larger models fail to saturate the DPU, and the smallest models are likely to drive it to an even lower utilization fraction.

The second is a measurement artifact arising from the 100% fallback estimation regime. If the single power sample captured per batch tends to fall during a lower-power phase of the inference window (such as the initial dispatch phase before DPU execution begins), the estimated power would be systematically below the true mean. This bias would be identical across all three repetitions of a model and would not appear in rep-to-rep CV.

Notably, the depth-3 models also show the largest positive EPI residuals relative to the global power law (mean +28% above the trend line). This initially seems contradictory. If power is underestimated, EPI should also be underestimated, producing negative residuals. The positive EPI residuals instead suggest that the nearest-sample estimator may correctly capture a higher-than-expected within-batch power spike for very fast inferences, even if it underestimates the mean across the full inference window. These competing effects make the depth-3 measurement interpretation genuinely ambiguous. Resolving it requires either higher-frequency power sampling or a controlled cross-validation experiment comparing estimator outputs against a reference measurement approach.

A practical consequence of this ambiguity is that the depth-3 EPI values should be treated with greater uncertainty than those of deeper models. If the measurement artifact explanation is correct, the true EPI of depth-3 models may be lower than reported, which would steepen the global EPI scaling exponent and shift the power law intercept. If the genuine DPU pipeline explanation is correct, the measured values reflect a real physical regime in which very short inferences engage the DPU differently, and the exponent remains an accurate description of on-device behavior. In either case, the depth-3 tier represents a qualitatively distinct operating regime,

and the depth-stratified GPR evaluation (Section 5.8.4) already treats it as such by reporting within-tier MAPE separately. Practitioners deploying models in the depth-3 range should be aware that the global GPR predictor is trained on a mixture of estimation regimes and that depth-3 EPI predictions carry an additional unquantified systematic uncertainty beyond the stated 95% prediction intervals.

6.4 Depth-Tier Patterns and the EPI Inversion

Two patterns in the results deserve specific discussion beyond the global scaling law.

The most striking anomaly in Table 5.3 is the EPI inversion in the depth-6 tier. Model-39m (38.8M parameters, 0% fallback) records a lower EPI (443,800 μJ) than model-26m (25.8M parameters, 56% mixed fallback, 485,000 μJ) despite having more parameters. This is inconsistent with the global power law and is most plausibly explained by the change in EPI estimator between these two models. Model-26m's mixed-fallback regime uses the nearest-sample approximation for roughly half of its batches. For a model at the fallback transition boundary, this approximation may overestimate EPI, as the nearest sample captures the power during the active DPU execution phase rather than the full within-batch average (which includes lower-power phases at the start and end of inference). Model-39m uses the fully trapezoidal estimator instead. The inversion is therefore most likely a measurement artifact of the regime transition, rather than a genuine non-monotonicity in EPI with parameter count.

The depth-6 tier also shows the highest within-tier EPI MAPE of any tier except depth-7 (approximately 54%), despite having five models. This heterogeneity has multiple independent sources, including the model-26m EPI inflation from the mixed estimator, model-11m's anomalously low power (7.88 W vs 8.11–8.27 W for its depth-6 neighbors), and the large EPI jump from model-39m to model-58m across a 1.5× increase in parameter count. The depth-6 MAPE of 54% reflects this compound variability rather than a fundamental failure of the GPR.

The three depth-7 models sit as a group above the global latency and EPI regression lines. For latency, positive residuals are 21%, 40%, and 67% for model-88m, model-132m, and model-200m respectively, and the magnitude grows with model size within the tier. One candidate explanation is that the very long inference times of depth-7 models (244–627 ms) expose latency components not well-captured by a global MAC-count regression, such as increased memory transfer time for the largest model weights. No confirmed mechanism has been identified, and the observation is reported as an open finding.

6.5 Why GPR Outperforms Simpler Models at N = 23

The GPR's dominance over linear, Ridge, and polynomial Ridge regression at $N = 23$ reflects a structural match between the model class and the problem. EPI and latency as functions of architecture parameters are smooth power laws across four orders of magnitude. Linear regression cannot represent this nonlinearity. Polynomial Ridge

introduces curvature but is prone to extrapolation instability with only 23 data points, particularly at the sparse depth-7 extreme where the polynomial fit must extrapolate far from the training centroid. GPR's squared-exponential ARD-RBF kernel approximates smooth power-law structure without requiring a parametric form assumption, and it adapts naturally to data density, with tighter predictions in the well-populated depth-4 through depth-6 range, and automatically wider uncertainty at the sparse depth-3 and depth-7 extremes.

The ARD property of the fitted kernels provides additional insight into the problem structure. For throughput, the GPR assigns a length scale of 0.60 to the depth-width interaction term and 3.35 to mean kernel size. A shorter length scale means the kernel predicts more rapid variation with that feature. The ratio of 5.6 between the two length scales shows that throughput is treated as varying much more strongly with the depth-width term than with kernel size, consistent with the feature selection result that kernel size is a weak secondary predictor. This automatic weighting is one of the practical advantages of GPR over fixed-form regressors, as the model learns the relative importance of features through kernel hyperparameter optimization, rather than having it imposed by the analyst.

The 95.7% empirical coverage of the stated 95% prediction intervals confirms coverage-consistent uncertainty within this benchmark, though calibration cannot be independently validated at $N = 23$. The widening of intervals for depth-6 and depth-7 models visible in Figure 5.18 is appropriate behavior for this application, as the GPR assigns greater uncertainty where fewer training examples exist near the test point. This calibration is well-behaved within the SimNet training distribution. A wide prediction interval signals the need for hardware validation before deployment decisions are finalized, while a narrow interval supports higher-confidence ranking. These coverage properties apply only within the benchmarked design space and verified compiler deployability range. Whether they extend to genuinely out-of-distribution architectures is unknown.

6.6 What the Features Reveal

The collapse of 29 candidate features into four clusters is as much a finding about the experimental design as it is about the features themselves. SimNet varies along a tightly parameterized grid. Given depth D and base channel width w, all other architectural quantities are essentially deterministic functions of those two values. The 25-feature Cluster 1 exists because the SimNet design constraints make all size-and-compute features co-monotone within the benchmarked set. A more heterogeneous model family would produce a substantially different cluster structure and might reveal independent predictive signal in features that are effectively redundant here.

The two singleton clusters reveal important properties of this design space. `arithmetic_intensity` achieves near-zero univariate Spearman correlation with EPI ($\rho = -0.09$) because all 23 SimNet models are deep in the compute-bound region with arithmetic intensities between 80 and 320 MACs/byte. There is simply very little variance in this quantity across the family to explain differences in EPI, even though arithmetic intensity is the central quantity in roofline analysis more generally.

`mean_kernel_size` has near-zero univariate correlation with EPI ($\rho = 0.12$) but

was selected as the second throughput predictor. This apparent contradiction resolves in the multivariate context. Once the depth-width term accounts for the dominant throughput variance, kernel size carries the remaining structured residual associated with per-layer DPU cycle count. The GPR then treats it as a weak secondary modulator (length scale 3.35 vs 0.60). This illustrates a general principle that low univariate correlation does not imply zero marginal predictive value when the primary predictor leaves structured residuals.

From a practical standpoint, the cluster collapse to four independent features means that the energy behavior of any SimNet model is captured by just two design-time numbers which are depth D and base channel width w. This is a strikingly compact characterization of a 29-dimensional feature space, and it is a direct consequence of the model family's constrained parameterization. For a practitioner exploring the SimNet design space, this means that EPI can be estimated before training simply by evaluating $D \times \log_{10}(w)$ and querying the GPR, provided the architecture lies within the verified deployable range or has passed pre-flight inspection. However, this compactness also marks the boundary of the predictor's transferability. A model family that uses residual connections, irregular kernel patterns, or attention modules would break the near-deterministic relationship between D, w, and all derived features. In that setting, features that are effectively redundant within SimNet (such as `receptive_field`, `channel_growth_ratio`, or `mac_per_param`) might carry independent predictive signal and the cluster structure would fragment into more than four groups. The feature engineering pipeline developed here is reusable, but the cluster assignments and the selected features should be treated as architecture-family-specific rather than universal.

6.7 The Compiler as a Hidden Deployment Risk

The CPU fallback in model-300m is a primary finding that extends beyond the scope of the predictive model. The key observation is that the SimNet architecture contains no depthwise-separable convolutions by design, yet the compiled binary for model-300m contains exactly such operators. The most direct interpretation is that the compiled graph represents standard convolutions as an operator decomposition at large channel widths. The decomposed operators exceed the DPUCZDX8G ISA boundary, causing CPU fallback. This should be read as an evidence-backed interpretation of the compiled graph rather than a proven description of compiler internals. The issue is that the channel-width threshold at which this transformation becomes ISA-incompatible is not published as a user-facing constraint, making the outcome not visible at the architecture design stage.

Model-200m, using the same depth-7 template with a base channel width of 808 (versus 992 for model-300m), compiles to a single DPU subgraph without fallback. The deployability threshold lies between base widths 808 and 992 for depth-7 architectures, yet the two models differ only in channel scale. This demonstrates that the deployability boundary is an empirical property of the compiler-hardware stack rather than a predictable consequence of the model architecture. Scaling laws and predictive models derived from the 23 deployable architectures cannot be assumed to extrapolate beyond the verified deployability range, because the boundary may be

crossed abruptly and without architectural warning.

The practical mitigation is the pre-flight inspection implemented in this study, which parses the compiled binary to verify that all operators are assigned to the DPU subgraph before benchmark time is invested. Without this check, a CPU-fallback model would reveal itself only at inference time through anomalously low throughput, with no obvious indication of the underlying cause. This finding suggests that deployment pipelines for DPU-based systems should treat empirical compilation verification as a required step, not an optional one. It also illustrates a broader point. The compiler toolchain is not a transparent pass-through from model design to hardware execution, and deployment-level energy characterization must account for transformations that design-level analysis cannot see.

6.8 Situating the Results

This work addresses all four research gaps identified in Section 3. The first gap concerned the lack of EPI as a primary, end-to-end deployment metric. This study demonstrates that EPI can be measured repeatably via on-board power sensors and predicted from architecture parameters with screening-level accuracy, providing a concrete methodology for practitioners who wish to treat EPI as a first-class criterion rather than a secondary observation.

The second gap concerned the absence of deployment-level energy prediction for DPU-based platforms, with existing approaches confined to the HLS or RTL design stage. The GPR predictors presented here operate entirely at deployment level. They require only post-training architecture parameters (and one compilation-derived feature) and capture real effects, including DPU dispatch overhead, the within-batch power profile, and power plateau behavior, that design-level models cannot represent. The compiler finding further confirms that design-level analysis is insufficient. The CPU fallback in model-300m is a compiler behavior that no pre-compilation energy model could anticipate.

The third gap concerned the sparse characterization of 1D CNN energy behavior on FPGA platforms. The SimNet benchmark provides a systematic sweep across four orders of magnitude in parameter count for the ZCU104 B4096 configuration. The finding that EPI scales as $P^{0.749}$ provides a concrete empirical reference point for future work, and the identification of the depth-width interaction term as the dominant EPI predictor offers a starting hypothesis for studies on other DPU configurations.

The fourth gap concerned the scarcity of public empirical datasets for FPGA inference energy. The sensor measurement data (raw per-batch power traces and per-repetition Parquet files) are released on Zenodo [10], the compiled `.xmodel` binaries on Zenodo [9], and the rubber bushing simulation dataset on Zenodo [8], all under permissive licenses, directly addressing this gap.

Taken together, the four gaps form a dependency chain. Public, well-documented datasets (gap 4) are the precondition for systematic characterization (gap 3). Characterization is the precondition for deployment-level predictive modeling (gap 2), and deployment-level prediction is what elevates EPI from a post-hoc observation to a first-class design criterion (gap 1). This study progresses along all four links of this chain for one specific configuration. The chain also clarifies what remains open. The

dataset addresses gap 4 for the ZCU104 B4096 under Vitis AI 3.5, but a dataset for even a single alternative configuration (a different DPU variant, a newer toolchain version, or a different model family) would need to be collected independently. Closing gap 4 in a configuration-agnostic sense requires a coordinated multi-platform effort of the kind that benchmark suites such as MLPerf Inference have undertaken for other deployment targets.

6.9 Scope and Limitations

The scaling exponents, feature selections, and predictive model reported here are specific to the ZCU104 B4096 platform, Vitis AI 3.5, and the SimNet 1D CNN family under batch-size-1 operation. The methodology itself is adaptable to other fixed hardware-toolchain configurations. The benchmarking pipeline, feature engineering approach, and GPR modeling procedure can be applied with new measurements on any target platform. The sensor measurement data, compiled model binaries, and simulation dataset are publicly released on Zenodo [8–10] to support replication and extension to other platforms and model families.

The SimNet family's homogeneity is a necessary condition for the clean scaling laws observed but limits the predictor's scope. The absence of skip connections, attention, depthwise operations, or irregular kernel patterns means that depth and channel width fully parameterize the family. A practitioner with a model that uses any of these features cannot directly apply the learned EPI mapping without retraining the predictor on measurements from that family. This is a scope limitation rather than a data sufficiency problem. The LOO-CV protocol already produces the available out-of-sample estimates for the SimNet family, subject to the partial-nesting caveat discussed in Section 6.1, and the 23 models represent the full deployable range of this family on this platform. Standard image classification architectures such as ResNet and MobileNet operate on 2D spatial data and were not benchmarked because they address a different input modality entirely. Established 1D CNN families such as temporal convolutional networks or dilated residual networks were not used because they vary along multiple architectural axes simultaneously, preventing clean attribution of energy differences to specific design parameters. Both of these are scope decisions grounded in experimental control rather than technical limitations of the platform. Extending the framework to heterogeneous 1D CNN families, and eventually to arbitrary CNN architectures, is the primary direction for future work and is discussed in Section 7.2.3.

Three measurement gaps from this study warrant priority in any follow-up. First, idle baseline measurements should be collected with the DPU loaded but not executing. Subtracting the idle baseline from inference measurements would isolate the DPU energy contribution, provide an independent check of the `pout1` estimates, and allow the ARM overhead component to be quantified. Second, `model-132m` and `model-200m` were benchmarked with randomly initialized weights rather than weights trained on the rubber bushing dataset. These two models constitute two of the three depth-7 models, the tier with the highest prediction error (EPI MAPE 68%) and the highest absolute EPI values in the family. Since activation distributions of random-weight models may differ from those produced by training, INT8 switching activity and hence

power may be systematically different from what would be observed with trained weights. This adds an unquantified systematic uncertainty to depth-7 predictions beyond the stated prediction intervals. Practitioners targeting architectures in this size range should prioritize hardware validation. A direct comparison of trained versus random-weight EPI for `model-88m` would bound this effect. Third, higher-frequency power sampling at fixed model size would characterize the within-batch power profile and resolve the ambiguity surrounding both the depth-3 anomaly and the trapezoidal EPI discrepancy described in Section 6.11.

6.10 Validity and Reliability

The results should be interpreted as deployment-level, system-level findings for the ZCU104 B4096 configuration rather than as pure DPU energy measurements or platform-independent scaling laws. The main measurement limitation is that the selected `pout1` rail covers the PS/PL power domain rather than the DPU fabric alone. As discussed in Section 6.11, this makes the reported EPI values the relevant system-level energy cost for deployment, but it also means that ARM-side scheduling, inference-loop execution, and sensor polling contribute an unquantified overhead component. The fixed 1.2 GHz CPU frequency and repeated thermal conditioning reduce this confound, but do not remove it.

A second measurement limitation is the use of dummy-input INT8 calibration. Since the benchmark targets energy, latency, and throughput rather than post-quantization classification accuracy, the main assumption is that calibration affects the absolute activation scale factors more than the relative cross-model energy ranking. This is plausible because the operation counts are architecture determined, but it was not directly validated by comparing dummy-calibrated and representative-calibrated models on hardware. The absolute EPI values should therefore be read with this systematic uncertainty in mind.

The predictive modeling results are also constrained by the size and structure of the deployable model set. The 23 benchmarked models represent the full SimNet family satisfying the single-DPU-subgraph constraint on this platform, but this is still a small dataset for regression. Leave-one-out cross-validation maximizes the training set per fold, but the estimates remain sensitive to individual architectures. In addition, feature selection is partially nested, so the reported prediction errors are lower-bound estimates rather than fully unbiased generalization estimates. The depth-3 regime analysis is exploratory for the same reason as the regime was identified after observing the data and should not be interpreted as a confirmed platform law without replication.

External validity is limited by the deliberate experimental control. The scaling exponents, selected features, and GPR predictors are specific to the ZCU104 B4096 platform, Vitis AI 3.5, batch-size of 1 execution, and the homogeneous SimNet 1D CNN family. This homogeneity is what makes clean attribution to depth and channel width possible, but it also means that the learned mapping should not be directly applied to architectures with residual connections, depthwise layers, attention mechanisms, or irregular kernel patterns without new measurements. Similarly, the use of simulated rubber-bushing data provides a controlled workload, but does not

establish practical anomaly-detection performance on real industrial sensor data.

Reliability is strengthened by the deterministic experimental design. Dataset generation, model construction, and parameter sampling use a fixed seed, the same 10,000 input windows are presented in the same order for every repetition, and each model is measured across three independently conditioned runs. The consistent preheat temperatures and reported across-repetition standard deviations provide direct evidence that the measurement pipeline is stable, even though the absolute energy values remain subject to the measurement limitations described above.

Table 6.1 summarizes the main methodological risks, the mitigations applied in this study, and the uncertainty that remains without new experiments.

Table 6.1: Methodological risks, mitigations, and remaining uncertainties.

Risk source	**Mitigation in this study**	**Remaining uncertainty**
System-level `pout1` rail	Fixed CPU frequency, identical benchmark loop, repeated thermal conditioning, and explicit interpretation as deployment-level energy.	ARM and PS/PL overhead are not baseline-subtracted, so pure DPU energy is not isolated.
Nearest-sample EPI estimator	Reported fallback rates and separated the fallback, transition, and trapezoidal regimes in the analysis.	Cross-regime comparisons near model-26m/model-39m may contain estimator bias.
Dummy INT8 calibration	Restricted claims to energy, latency, and throughput rather than post-quantization accuracy, and reported the assumption explicitly.	Real-data calibration was not benchmarked, so model-specific calibration effects remain unquantified.
Random weights in depth-7 models	Marked random-weight models and interpreted depth-7 results cautiously.	Depth-7 activation statistics may differ from trained models.
Partially nested feature selection	Reported prediction errors as lower-bound estimates and emphasized MdAPE/log-MAPE alongside MAPE.	Fully nested selection may produce higher errors, especially for the sparse depth-7 region.
CPU fallback	Applied pre-flight subgraph inspection and excluded fallback models before benchmarking.	The exact compiler threshold and mechanism remain toolchain-specific and undocumented.

6.11 Measurement Validity

The nearest-sample fallback estimator introduces an unquantified uncertainty for all models in the 100% fallback tier. As shown in Section 5.4, this tier extends to models with inference times up to 34.6 ms because OS scheduling jitter on the polling thread routinely produces zero interior samples even at latencies well above the nominal 20 ms polling interval. The consistently low rep-to-rep CV for these models (below

0.5% for depth-3 and depth-4) provides indirect evidence that measured power is stable within individual inferences, making the nearest sample a reasonable proxy for the mean. A systematic bias, however, cannot be ruled out, and any absolute comparison of EPI across the fallback and trapezoidal tiers should be made with this caveat in mind. Relative comparisons within the fallback tier are less affected because any systematic bias applies consistently across all fallback-tier models.

For the fully trapezoidal models (0% fallback), the `pout1` EPI derived from the trapezoidal integral is systematically lower than the product of average power and latency. The ratio between the two grows from approximately 1.1 for model-200m to over 2.0 for model-39m. This is not a data error. It reflects a genuine physical characteristic. The DPU does not draw constant power throughout the inference window. It ramps from a lower-power dispatch state, executes the computation, and returns to idle, producing a within-batch power profile that integrates to less than peak power times duration. The trapezoidal integral captures this lower integrated energy, while the average power column reflects instantaneous sensor readings that include the peak execution phase. The `pout1` EPI values are therefore the energy metric used in this study, rather than a simple power-times-latency estimate.

The choice to use `pout1` rather than a DPU-isolated rail is deliberate. For an embedded deployment scenario, the system-level energy per inference is the quantity that determines battery life and thermal budget. A DPU-only energy measurement would require a platform with an accelerator-isolated rail, which the ZCU104 does not provide. A potential confound is that ARM energy scales with latency ($E_{\text{ARM}} = P_{\text{ARM}} \times T$). The ratio across `model-25k` to `model-200m` is approximately 770×. During inference the ARM cores run the Python inference loop, VART scheduler, and sensor thread. Active dynamic power at 1.2 GHz under this workload is substantially higher than idle power, and no published characterization of this workload-specific ARM power was found. The ARM energy confound is therefore unquantified in absolute terms. Cross-model comparisons remain informative because the ARM-side workload (benchmark loop and sensor polling) was identical across all 23 models, so ARM power P_{ARM} is plausibly similar across runs under fixed-frequency operation. This assumption was not validated with an idle or no-inference baseline. ARM energy per inference is $E_{\text{ARM}} = P_{\text{ARM}} \times T$, so any ARM component contributes to the reported system-level EPI and prevents DPU-only energy attribution. The relative EPI rankings should therefore be interpreted as system-level rankings for this deployment stack rather than isolated DPU energy rankings.

6.12 Ethical, Societal, and Sustainability Aspects

6.12.1 Sustainability

EPI, as defined and measured here, is a direct measure of the energy cost of a single inference. Aggregated over the continuous operation of an embedded system, small differences in EPI between candidate architectures translate into meaningful differences in energy consumption and, by extension, in carbon footprint. The predictive framework developed in this study contributes to reducing the cost of finding low-energy architectures by narrowing the candidate set through EPI prediction

before hardware benchmarking. Practitioners can make deployment decisions with fewer physical experiments, each of which itself consumes energy and time. The power law $E \propto P^{0.749}$ provides a concrete quantitative basis for this trade-off. Moving from a 10M-parameter model to a 1M-parameter model reduces EPI by a factor of approximately $5\times$, a reduction that a practitioner can now estimate without running the hardware. The compiler deployability finding adds a further dimension. Benchmark campaigns on non-deployable architectures represent energy and resource expenditure that the pre-flight inspection step can prevent before it is incurred.

6.12.2 Societal Impact

The framework lowers a barrier to energy-efficient AI adoption on constrained hardware. Embedded and edge deployments are increasingly relevant in contexts where cloud infrastructure is either too costly, too energy-intensive, or unavailable, including industrial sensing, environmental monitoring, and distributed robotics. Making EPI prediction accessible from architecture parameters alone, without requiring repeated hardware access, reduces the specialization required to reason about energy trade-offs in these settings. Smaller organizations and research groups that lack dedicated FPGA infrastructure can now use the public dataset and predictive model to make informed architecture choices before committing to hardware procurement. The throughput results also have a direct societal reading. The finding that all models up to and including `model-5_1m` (5.1M parameters) meet the 50 Hz real-time threshold provides concrete guidance for practitioners deploying anomaly detection in safety-critical industrial monitoring applications.

6.12.3 Ethical Considerations

The public release of the benchmark dataset and compiled model binaries on Zenodo reflects a commitment to research transparency and replicability. All exclusion decisions, measurement limitations, and exploratory analyses are reported and acknowledged in this thesis. The partial nesting of feature selection and the lower-bound status of the reported MAPEs are stated explicitly rather than glossed over, and no data or results have been selectively withheld.

The framework itself carries no direct risk of harm in its current form, as it processes only physics-based simulation data and architectural parameters rather than any personal or sensitive information. The released artifacts and methodology could, in principle, be used to characterize energy consumption in other CNN-based inference systems, including applications with higher ethical stakes such as surveillance or decision-support systems. The transparency of the methodology provides a foundation for such applications to be evaluated critically, but the ethical implications of the specific application remain the responsibility of practitioners who extend the framework beyond its current scope.

Chapter 7

Conclusions and Future Work

7.1 Conclusions

Deploying a CNN on an FPGA for real-time anomaly detection requires choosing among architectures that differ substantially in energy cost, throughput, and latency. This thesis investigated whether energy-per-inference (EPI) can be predicted from model architecture parameters alone, before hardware benchmarking begins, so that hardware access can be reserved for validating a small final shortlist.

The study benchmarked 23 deployable 1D CNN models spanning four orders of magnitude in parameter count on the ZCU104 with a B4096 DPU, producing a 690,000-row per-batch dataset. Gaussian process regression (GPR) predictors were trained on architecture features and evaluated under leave-one-out cross-validation.

7.1.1 RQ1: Estimating EPI, Latency, and Throughput

Each target is predicted by a GPR trained on one or two features selected by forward LOO-CV from a de-collinearized pool of 29 candidate features. For EPI, the sole selected feature is the engineered interaction term `depth_x_log_width` ($= D \times \log_{10} w$), which outperformed the more obvious `log10_params` because parameter count conflates depth and channel width across depth tiers while the interaction term encodes their joint effect on measured power draw. Latency is predicted from `log10_params` alone, and throughput uses `depth_x_log_width` with `mean_kernel_size` as a second feature. Before any EPI prediction is meaningful, deployability must be verified by pre-flight subgraph inspection, as architectures with CPU fallback fall outside the predictor's valid range.

7.1.2 RQ2: Architecture and Energy Relationships

EPI follows a power law ($E \propto P^{0.749}$, $R^2 = 0.984$) across four orders of magnitude. The sublinear exponent of 0.749 follows from sublinear latency scaling together with the measured power plateau. Latency itself follows a power law with exponent 0.755, while measured power plateaus at approximately 8.2 W for depth-6 and depth-7 models after rising from 5.5 W at depth-3. This measured power plateau is consistent with a structural mismatch between the DPU's spatial tiling architecture, optimized for the large feature maps of image CNNs, and SimNet's progressive pooling strategy, which reduces the 512-sample input to 32 elements or fewer by the third or fourth layer. A previously undocumented compiler deployability boundary was also found.

`model-300m` and `model-200m` share an identical depth-7 template and differ only in base channel width (992 vs 808), yet exactly one compiles to a single DPU subgraph. Inspection of the compiled binaries indicates that large-channel-width convolutions can be represented as depthwise-separable operators at an undocumented threshold, causing ISA boundary violations that are not evident before compilation is attempted.

7.1.3 RQ3: Prediction Accuracy

The GPR achieves lower-bound MdAPEs of 13.8%, 12.6%, and 15.4% for EPI, latency, and throughput respectively, and log-MAPEs of 2.09% and 2.59% for EPI and latency. All MAPEs are lower bounds due to partial nesting of feature selection. Empirical coverage of the stated 95% prediction intervals is 95.7%, consistent with the nominal level at $N = 23$. The predictor is well-suited to architecture screening and ranking, but should not replace hardware measurement for precise power budgeting.

7.1.4 Contributions

This thesis makes three distinct contributions. First, it identifies and documents the Vitis AI compiler deployability boundary as a primary engineering constraint, and documents the `benchmark-inspect` pre-flight procedure used to detect CPU fallback subgraphs before benchmarking time is committed. Second, it provides a transparent and replicable methodology for deployment-level EPI prediction for CNNs on DPU platforms. Third, it releases the first publicly available empirical dataset of deployment-level FPGA inference energy for 1D CNN models, comprising raw per-batch power traces [10], compiled `.xmodel` binaries [9], and the rubber bushing simulation dataset [8], all deposited on Zenodo under permissive licenses.

7.2 Future Work

7.2.1 Measurement Improvements

Three measurement gaps warrant priority. Running the board with the DPU loaded but not executing would provide an idle baseline that, when subtracted from inference measurements, approximates the workload-dependent energy contribution and quantifies the ARM overhead that is currently uncharacterized. Comparing EPI under dummy-calibrated versus real-data-calibrated INT8 quantization for at least one model would bound the systematic uncertainty affecting all absolute EPI values. Higher-frequency power sampling, using the on-board ADC or an external current probe, would resolve the within-batch power profile and clarify both the depth-3 anomaly and the trapezoidal EPI discrepancy.

7.2.2 Extending to Other Platforms

All results are specific to the ZCU104 B4096 under Vitis AI 3.5. On the ZCU104, alternative DPU configurations (B2048, B1024, B512) require only a different bitstream overlay, making intra-platform extension straightforward. Extending to other

boards (ZCU102, Kria KV260, Alveo U50) or edge accelerators (Coral Edge TPU, Hailo-8, NVIDIA Jetson Orin) would test the generalizability of the scaling laws and the compiler deployability boundary across the wider embedded AI landscape.

7.2.3 A More General Predictor

The most immediate step is extending beyond the homogeneous SimNet family to architectures with residual connections, depthwise separable convolutions, or attention modules. Such a heterogeneous dataset would reveal which of the 29 candidate features carry independent predictive signal when depth and channel width no longer fully parameterize the architecture. The long-term goal is a tool that accepts an arbitrary PyTorch model, verifies deployability via subgraph inspection, and returns GPR-predicted EPI with model-based prediction intervals before any hardware access is required. The datasets released on Zenodo [8–10] and the benchmarking pipeline provide the starting point for this effort.

References

[1] AMD/Xilinx. (2023) Deep Learning Processing Unit (DPU) IP Product Guide (PG338). [Online]. Available: https://docs.xilinx.com/r/en-US/pg338-dpu

[2] ——. (2023) PYNQ — Python Productivity for Zynq. [Online]. Available: http://www.pynq.io/

[3] ——, *Vitis AI User Guide (UG1414)*, AMD/Xilinx, 2023. [Online]. Available: https://docs.xilinx.com/r/en-US/ug1414-vitis-ai

[4] ——. (2023) ZCU104 Evaluation Board. [Online]. Available: https://www.xilinx.com/products/boards-and-kits/zcu104.html

[5] C. M. Bishop, *Pattern Recognition and Machine Learning*. Springer, 2006.

[6] M. Blott, T. B. Preußer, N. J. Fraser, G. Gambardella, K. O'Brien, Y. Umuroglu, M. Leeser, and K. Vissers, "FINN-R: An end-to-end deep-learning framework for fast exploration of quantized neural networks," *ACM Transactions on Reconfigurable Technology and Systems*, vol. 11, no. 3, pp. 1–23, 2018. [Online]. Available: https://doi.org/10.1145/3242897

[7] X. Chen *et al.*, "Atapp: Architecture- and technology-aware power prediction for fpgas," in *ATAPP: Architecture- and Technology-Aware Power Prediction for FPGAs*, 2025. [Online]. Available: https://lca.ece.utexas.edu/pubs/FPL_2025_ATAPP.pdf

[8] M. Fredriksson, "Rubber Bushing Simulation Data," 2026. [Online]. Available: https://doi.org/10.5281/zenodo.19846170

[9] ——, "SimNet Compiled FPGA Models," 2026. [Online]. Available: https://doi.org/10.5281/zenodo.19847357

[10] ——, "ZCU104 DPU Inference Benchmark Dataset," 2026. [Online]. Available: https://doi.org/10.5281/zenodo.19763074

[11] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016. [Online]. Available: https://www.deeplearningbook.org

[12] B. Jacob, S. Kligys, B. Chen, M. Zhu, M. Tang, A. Howard, H. Adam, and D. Kalenichenko, "Quantization and training of neural networks for efficient integer-arithmetic-only inference," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018, pp. 2704–2713. [Online]. Available: https://doi.org/10.1109/CVPR.2018.00286

[13] R. Kedia, S. Goel, M. Balakrishnan, K. Paul, and R. Sen, "Design space exploration of fpga-based system with multiple dnn accelerators," *IEEE*

Embedded Systems Letters, vol. 13, no. 3, pp. 114–117, 2021. [Online]. Available: https://ieeexplore.ieee.org/document/9170518

[14] S. Khandelwal, A. Walsh, and S. Shreejith, "Quantised neural network accelerators for low-power ids in automotive networks," in *2023 Design, Automation amp; Test in Europe Conference amp; Exhibition (DATE)*. IEEE, Apr. 2023, p. 1–2. [Online]. Available: https://ieeexplore.ieee.org/document/10137016

[15] G. Lacey, G. W. Taylor, and S. Afshar, "Deep learning on FPGAs: Past, present, and future," *arXiv preprint arXiv:1602.04283*, 2016. [Online]. Available: https://arxiv.org/abs/1602.04283

[16] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, pp. 436–444, 2015. [Online]. Available: https://doi.org/10.1038/nature14539

[17] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[18] Z. Lin, J. Zhao, S. Sinha, and W. Zhang, "Hl-pow: A learning-based power modeling framework for high-level synthesis," in *2020 25th Asia and South Pacific Design Automation Conference (ASP-DAC)*, 2020, pp. 574–580. [Online]. Available: https://ieeexplore.ieee.org/document/9045442

[19] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Köpf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, "PyTorch: An imperative style, high-performance deep learning library," in *Advances in Neural Information Processing Systems 32 (NeurIPS)*, 2019, pp. 8024–8035. [Online]. Available: https://arxiv.org/abs/1912.01703

[20] PyTorch Contributors, "AdamW," https://docs.pytorch.org/docs/stable/generated/torch.optim.AdamW.html, 2026, pyTorch documentation. Accessed: 22 May 2026.

[21] ——, "CosineAnnealingLR," https://docs.pytorch.org/docs/stable/generated/torch.optim.lr_scheduler.CosineAnnealingLR.html, 2026, pyTorch documentation. Accessed: 22 May 2026.

[22] C. E. Rasmussen and C. K. I. Williams, *Gaussian Processes for Machine Learning*. MIT Press, 2006, ISBN: 026218253X.

[23] Y. R. M. Reddy, P. Muralidhar, G. Satya Narayana, and D. Jagan, "Fpga(zcu104) based energy efficient accelerator for mobilenet-v1," in *2024 20th IEEE International Colloquium on Signal Processing Its Applications (CSPA)*, 2024, pp. 57–62. [Online]. Available: https://ieeexplore.ieee.org/document/10525375

[24] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," *Nature*, vol. 323, pp. 533–536, 1986.

[25] Z. Wei, A. Arora, E. Shriver, and L. John, "Cross-fpga power estimation from high level synthesis via transfer-learning," in *Proceedings of the 2024 ACM/SIGDA International Symposium on Field Programmable Gate Arrays*, ser.

FPGA '24. New York, NY, USA: Association for Computing Machinery, 2024, p. 187. [Online]. Available: https://dl.acm.org/doi/10.1145/3626202.3637621

[26] Wikipedia contributors. (2024) Friction — Coulomb friction. Accessed: 29 April 2026. [Online]. Available: https://en.wikipedia.org/wiki/Friction

[27] ——. (2024) Generalized Maxwell model. Accessed: 29 April 2026. [Online]. Available: https://en.wikipedia.org/wiki/Generalized_Maxwell_model

[28] ——. (2024) Viscoelasticity. Accessed: 29 April 2026. [Online]. Available: https://en.wikipedia.org/wiki/Viscoelasticity

[29] S. Williams, A. Waterman, and D. Patterson, "Roofline: An insightful visual performance model for multicore architectures," *Communications of the ACM*, vol. 52, no. 4, pp. 65–76, 2009. [Online]. Available: https://doi.org/10.1145/1498765.1498785

[30] G. Özdil and B. Örs, "Model-based fpga ai accelerator design using vitis ai with in-depth performance and energy efficiency analysis," in *2024 32nd Telecommunications Forum (TELFOR)*, 2024. [Online]. Available: https://ieeexplore.ieee.org/document/10819065

Appendix A

Supplemental Information

A.1 SimNet Model Family

Table A.1 lists all 26 models in the SimNet benchmark family. The 24 size-scaled models are assigned to depth tiers in blocks of five (four for the deepest tier), with width chosen via binary search to match logarithmically spaced parameter targets. The two stress-probe models (boundary and overflow) are hand-constructed at depth 7 to target specific DPU memory thresholds and are not trained. The *DPU-only* column indicates whether the compiled model passed the pre-flight subgraph inspection described in Section 4.4. Models marked × in this column contain CPU fallback subgraphs and are excluded from the benchmark dataset. `model-200m` (base channel width 808) compiles cleanly, while `model-300m` (base channel width 992) does not, establishing that the deployability threshold lies between base widths 808 and 992 for depth-7 architectures. Depth gives the number of convolutional layers. *Trained*: model was trained on the rubber bushing dataset (✓) or uses random weights (×). *Compiled*: successful `vai_c_xir` compilation. *DPU-only*: all compute executes on the DPU with no CPU fallback subgraphs (✓) or excluded due to CPU fallback/compilation failure (×). Stress probes are separated by a horizontal rule. [a] *DPU-only*: all compute executes on the DPU with no ARM CPU fallback subgraphs detected during pre-flight inspection.

Table A.1: SimNet benchmark model family.

Model	Parameters	Depth	Trained	Compiled	DPU-only[a]
model-25k	25,385	3	✓	✓	✓
model-39k	39,385	3	✓	✓	✓
model-59k	58,713	3	✓	✓	✓
model-85k	84,521	3	✓	✓	✓
model-129k	128,545	3	✓	✓	✓
model-198k	197,705	4	✓	✓	✓
model-282k	282,169	4	✓	✓	✓
model-440k	439,545	4	✓	✓	✓
model-658k	657,921	4	✓	✓	✓
model-973k	972,537	4	✓	✓	✓
model-1_5m	1,473,537	5	✓	✓	✓
model-2_2m	2,229,361	5	✓	✓	✓
model-3_4m	3,378,601	5	✓	✓	✓
model-5_1m	5,051,921	5	✓	✓	✓
model-7_6m	7,583,625	5	✓	✓	✓
model-11m	11,398,681	6	✓	✓	✓
model-17m	17,283,305	6	✓	✓	✓
model-26m	25,840,833	6	✓	✓	✓
model-39m	38,802,553	6	✓	✓	✓
model-58m	58,406,369	6	✓	✓	✓
model-88m	88,211,025	7	✓	✓	✓
model-132m	132,307,793	7	×	✓	✓
model-200m	199,565,105	7	×	✓	✓
model-300m	300,315,777	7	×	✓	×
model-boundary	510,415,073	7	×	✓	×
model-overflow	580,070,521	7	×	×	×

A.2 Feature Definitions

Table A.2 lists all 29 features in the candidate pool used for predictive modeling (Section 4.7). Features marked **R** are read directly from the model configuration. Features marked **D** are derived by `compute_architecture_features()`. Features marked **C** are obtained from the compiled `.xmodel` artifact (requiring the Vitis AI compilation step but no on-device measurement). The four Spearman-clustering groups identified in Section 4.7 are indicated in the Cluster column. **S** = size-and-compute cluster, **G** = pooling-and-kernel-geometry cluster, **K** = mean-kernel-size singleton, **A** = arithmetic-intensity singleton, **X** = engineered interaction term evaluated directly in forward selection alongside cluster representatives.

Feature	**Type**	**Cluster**	**Definition**
`param_count`	R	S	Total trainable parameter count $P = \sum_i \lvert\theta_i\rvert$
`depth`	R	S	Number of Conv1d layers $D \in \{3, 4, 5, 6, 7\}$

Feature	Type	Cluster	Definition
base_width	R	S	Output channels of the first layer. Sets the channel ladder scale
total_macs	R	S	Total multiply-accumulate operations $M = \sum_i C_{\text{in},i} C_{\text{out},i} k_i L_i$
dpu_op_count	C	S	Number of operator nodes in the DPU subgraph of the compiled .xmodel. Depth-keyed $\{3 \to 48,\ 4 \to 56,\ 5 \to 64,\ 6 \to 72,\ 7 \to 80\}$. Empirically observed to satisfy dpu_op_count $= 40 + 8 \times D$ on this platform after compilation, making it a linear re-scaling of depth. Treated as compilation-derived because this relationship was not known prior to measurement and may not generalize to other DPU configurations
log10_params	D	S	$\log_{10}(P)$
log10_macs	D	S	$\log_{10}(M)$
log10_width	D	S	$\log_{10}(w)$
log10_params_sq	D	S	$(\log_{10}(P))^2$. Captures curvature in log-scale regressions
params_per_layer	D	S	P/D. Mean parameter count per convolutional layer
log10_params_per_layer	D	S	$\log_{10}(P/D)$
mac_per_param	D	S	M/P. Reuse factor and average MACs per parameter
compute_to_memory_ratio	D	S	M/P. Numerically identical to mac_per_param, retained as an independent code-path audit cross-check to verify consistency between the two derivation pipelines. Both are included in the 29-feature candidate pool but the clustering step will place them in the same cluster and only one will be selected as cluster representative
macs_per_dpu_op	D	S	M/n_{op}. Mean compute per compiled DPU instruction
params_per_dpu_op	D	S	P/n_{op}. Mean parameters touched per compiled DPU instruction
n_regular_pool_ops	D	G	$\lfloor D/2 \rfloor$. MaxPool1d(2) operations inserted between conv blocks
n_trailing_pool_ops	D	G	$\lfloor \log_2 L_{\text{after}} \rfloor$ where $L_{\text{after}} = 512/2^{n_{\text{regular}}}$. Trailing pools reducing feature map to length 1
final_seq_len_before_trailing	D	G	$\max(1,\ 512/2^{n_{\text{regular}}})$. Sequence length entering the trailing pool chain

Feature	Type	Cluster	Definition
`mean_kernel_size`	D	K	$\frac{1}{D}\sum_i k_i$. Arithmetic mean of per-layer kernel sizes
`max_kernel_size`	D	G	$\max_i k_i$. Largest kernel in the network (always the first layer)
`total_kernel_size`	D	S	$\sum_i k_i$. Sum of all per-layer kernel sizes
`receptive_field`	D	S	Effective input receptive field $\sum_i k_i \prod_{j<i} s_j$ where $s_j = 2$ at each regular pool, $s_j = 1$ otherwise
`first_layer_channels`	D	S	$C_{\text{out},1}$. Output channels of the first Conv1d layer
`last_layer_channels`	D	S	$C_{\text{out},D}$. Output channels of the final Conv1d layer
`max_channels`	D	S	$\max_i C_{\text{out},i}$. Equals `last_layer_channels` in this monotonically-expanding family
`channel_growth_ratio`	D	G	$C_{\text{out},D}/C_{\text{out},1}$. Ratio of last to first layer channel width
`total_activations`	D	S	$\sum_i C_{\text{out},i} \cdot L_i$. Total activation tensor elements across all layers
`log10_total_activations`	D	S	$\log_{10}$(total_activations)
`arithmetic_intensity`	D	A	$M/(P$ + total_activations). Roofline compute-per-byte ratio assuming INT8 (1 byte per element)
`depth_x_log_width`	D	X	$D \times \log_{10}(w)$. Interaction term encoding the joint effect of depth and channel scale. Evaluated in forward selection alongside the Cluster S representative (`log10_params`) by LOO-CV MAPE. It outperformed `log10_params` on EPI MAPE and selected as the primary EPI predictor and first-selected throughput feature. See Section 6.1.

Table A.2: Architecture feature candidate pool used for predictive modeling. P = parameter count, D = depth, w = base channel width, $C_{\text{out},i}$ = output channels of layer i, k_i = kernel size of layer i, L_i = sequence length at layer i, M = total MACs, n_{op} = DPU op count.

A.3 Simulation Parameter Ranges

Table A.3 lists the per-family physical parameters of the rubber bushing simulation and their sampling ranges. For each of the 100 families, all parameters are drawn independently and uniformly from the ranges shown. The parameters define the mechanical operating point of the simulated rubber bushing. The excitation geometry (A, f, offset), the nonlinear elastic behavior (k_0, k_3, x_{prog}), the viscoelastic relaxation

dynamics (two Maxwell branches with stiffnesses $k_{i,1}$, $k_{i,2}$ and time constants τ_1, τ_2), and the friction characteristics (F_c, v_s, c_v). Sensor noise levels (σ_x, σ_F) and an optional low-pass filter (applied in one third of families, with cutoff frequency drawn uniformly from 40–120 Hz) are also varied to ensure the model is exposed to a realistic range of measurement conditions.

Within each family, a ±7% perturbation is applied independently across the 30 simulation runs to a subset of parameters consisting of excitation amplitude A, excitation frequency f, linear stiffness k_0, Coulomb friction force F_c, and viscous damping coefficient c_v. All other parameters remain fixed at their family-level draw for all 30 runs.

Table A.3: Per-family simulation parameters and their sampling ranges. Each parameter is drawn independently and uniformly from the stated range for each of the 100 families. Parameters marked with † are additionally perturbed by ±7% across the 30 runs within each family.

Parameter	Symbol	Range	Unit
Excitation amplitude†	A	7–15	mm
Excitation frequency†	f	1–3	Hz
Static offset	offset	±2	mm
Linear stiffness†	k_0	18 000–32 000	N/m
Cubic stiffening	k_3	$2–5 \times 10^{11}$	N/m^3
Progressive onset	x_{prog}	2–10	mm
Fast Maxwell stiffness	$k_{i,1}$	$7–14 \times 10^5$	N/m
Slow Maxwell stiffness	$k_{i,2}$	$3.5–8 \times 10^5$	N/m
Fast relaxation time	τ_1	0.03–0.08	s
Slow relaxation time	τ_2	0.25–0.70	s
Coulomb friction†	F_c	35–70	N
Friction transition velocity	v_s	5×10^{-5}–5×10^{-4}	m/s
Viscous damping†	c_v	150–300	N·s/m
Displacement noise std.	σ_x	$0–5 \times 10^{-4}$	m
Force noise std.	σ_F	0–2	N

Faculty of Computing, Blekinge Institute of Technology, 371 79 Karlskrona, Sweden