# QUIDAM: A Framework for Quantization-Aware DNN Accelerator and Model Co-Exploration

AHMET INCI, Carnegie Mellon University, USA
SIRI GARUDANAGIRI VIRUPAKSHA, Carnegie Mellon University, USA
AMAN JAIN, Carnegie Mellon University, USA
TING-WU CHIN, Carnegie Mellon University, USA
VENKATA VIVEK THALLAM, Carnegie Mellon University, USA
RUIZHOU DING, Carnegie Mellon University, USA
DIANA MARCULESCU, Carnegie Mellon University, The University of Texas at Austin, USA

As the machine learning and systems communities strive to achieve higher energy-efficiency through custom deep neural network (DNN) accelerators, varied precision or quantization levels, and model compression techniques, there is a need for design space exploration frameworks that incorporate quantization-aware processing elements into the accelerator design space while having accurate and fast power, performance, and area models. In this work, we present *QUIDAM*, a highly parameterized quantization-aware DNN accelerator and model co-exploration framework. Our framework can facilitate future research on design space exploration of DNN accelerators for various design choices such as bit precision, processing element type, scratchpad sizes of processing elements, global buffer size, number of total processing elements, and DNN configurations. Our results show that different bit precisions and processing element types lead to significant differences in terms of performance per area and energy. Specifically, our framework identifies a wide range of design points where performance per area and energy varies more than 5× and 35×, respectively. With the proposed framework, we show that lightweight processing elements achieve on par accuracy results and up to 5.7× more performance per area and energy improvement when compared to the best INT16 based implementation. Finally, due to the efficiency of the pre-characterized power, performance, and area models, QUIDAM can speed up the design exploration process by 3-4 orders of magnitude as it removes the need for expensive synthesis and characterization of each design.

## 1 INTRODUCTION

Deep neural networks (DNNs) have achieved remarkable accomplishments across various applications ranging from image recognition [47], object detection [48], to natural language processing [6]. However, the increasing model size and computational cost of these models has become a challenging task for on-device machine learning (ML) endeavours due to the stringent performance per area and energy constraints of the edge devices. To this end, while machine learning practitioners focus on model compression techniques [5, 8, 15], computer architects

Authors' addresses: Ahmet Inci, ainci@andrew.cmu.edu, Carnegie Mellon University, 5000 Forbes Avenue, Pittsburgh, PA, USA, 15213; Siri Garudanagiri Virupaksha, Carnegie Mellon University, 5000 Forbes Avenue, Pittsburgh, PA, USA, 15213; Aman Jain, Carnegie Mellon University, 5000 Forbes Avenue, Pittsburgh, PA, USA, 15213; Ting-Wu Chin, Carnegie Mellon University, 5000 Forbes Avenue, Pittsburgh, PA, USA; Venkata Vivek Thallam, Carnegie Mellon University, 5000 Forbes Avenue, Pittsburgh, PA, USA; Ruizhou Ding, Carnegie Mellon University, 5000 Forbes Avenue, Pittsburgh, PA, USA; Diana Marculescu, dianam@utexas.edu, Carnegie Mellon University,, 5000 Forbes Avenue, Pittsburgh, PA, 15213 and The University of Texas at Austin, 110 Inner Campus Drive, Austin, TX, USA, 78705.

1539-9087/2022/9-ART $15.00
https://doi.org/10.1145/3555807

investigate hardware architectures to overcome the energy-efficiency problem and improve the overall system performance [2, 3, 14, 18–25, 39, 42].

As computing community hits the limits on consistent performance scaling for traditional architectures, there has been a rising interest on enabling on-device machine learning applications through custom domain-specific DNN accelerators. These domain-specific system-on-chip architectures are specifically designed to exploit the application characteristics. As we deeply care about performance per area and energy-efficiency from a hardware point of view, tailored DNN accelerators have shown significant improvements when compared to general-purpose CPUs and GPUs [2, 10, 27–29, 37]. To better understand the trade-offs of various architectural design choices and DNN workloads for domain-specific hardware architectures, there is a need for a design space co-exploration framework that can rapidly iterate over various designs and generate power, performance, and area (PPA) results for various DNN models. To this end, in this work we present *QUIDAM*, a framework for quantization-aware DNN accelerator and model co-exploration.

This work makes the following contributions:

- We present *QUIDAM*, a quantization-aware power, performance, and area modeling framework for DNN accelerators. Our framework can enable future research on DNN accelerator and model co-exploration for various design choices such as bit precision, processing element types, scratchpad sizes of processing elements, global buffer size, device bandwidth, number of total processing elements, and DNN architectures.
- Our framework provides power, performance, and area results not just for a single hardware design point but for a range of different hardware designs as opposed to prior art [1, 2, 4, 31, 38]. Thus, it can be used to jointly analyze trade-offs of various architectural design choices and DNN workloads and perform multi-objective optimization to achieve a better trade-off front between accuracy and hardware-efficiency metrics such as performance per area and energy.

The rest of the paper is organized as follows. In Section 2, we present a literature review on power and runtime models for CNNs and design space exploration frameworks for hardware accelerators. In Section 3, we describe the architectural details of the *QUIDAM* framework and the details of our methodology for power, performance, and area modeling of DNN accelerators. In Section 4, we show experimental results demonstrating the efficacy of *QUIDAM*'s power, performance, and area models and the efficacy of lightweight processing elements to conventional designs in terms of performance per area and energy through a suite of case studies. Finally, Section 5 concludes the paper by summarizing the results.

## 2 RELATED WORK

Prior art has proposed runtime and energy models for DNN workloads [1, 34, 38]. However, these models have been implemented specifically for GPU platforms and thus they create an important limitation for a design space exploration of hardware architectures and potentially hardware and machine learning model co-design opportunities [13, 51, 53].

On the other hand, the systems community has proposed several tools and simulation methodologies for DNN accelerator design. Table 1 shows a comparison of existing hardware accelerator frameworks compared to *QUIDAM* in terms of various design features in chronological order.

For example, Aladdin [43] is a pre-RTL power, performance, and area estimation tool for arbitrary hardware accelerators. This simulator takes high-level language descriptions of algorithms as inputs similar to a high-level synthesis (HLS) methodology and generates dynamic data dependence graphs as an approximate representation of a hardware accelerator without generating RTL. While this approach can be useful for fast exploration of various algorithms, it has limitations in the optimization of the generated hardware accelerators for deep neural networks because it is not implemented in a domain-specific architecture manner.

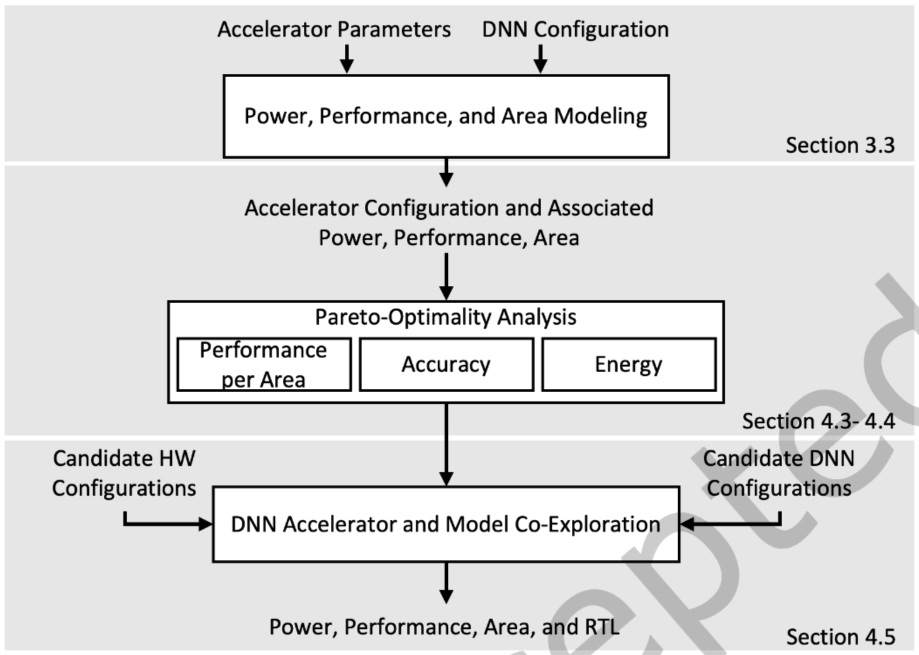

Accelerator Parameters
DNN Configuration
Power, Performance, and Area Modeling
Section 3.3
Accelerator Configuration and Associated Power, Performance, Area
Pareto-Optimality Analysis
Performance per Area
Accuracy
Energy
Section 4.3- 4.4
Candidate HW Configurations
Candidate DNN Configurations
DNN Accelerator and Model Co-Exploration
Power, Performance, Area, and RTL
Section 4.5

Fig. 1. Overview of the *QUIDAM* framework.

Table 1. Comparison of DNN Accelerator Frameworks

| | Optimized for CNNs | Fully-Parameterized RTL | Row-Stationary Dataflow | Quantization Support | Lightweight PE (Shift-Add Based) | Pre-Characterized Model based DSE | DNN Accelerator/ Model Co-Exploration |
|---|---|---|---|---|---|---|---|
| Aladdin [43] | ✗ | ✗ | ✗ | ✗ | ✗ | ✓ | ✗ |
| SCALE-Sim [40] | ✓ | ✗ | ✗ | ✗ | ✗ | ✓ | ✗ |
| Eyeriss [2] | ✓ | ✗ | ✓ | ✗ | ✗ | ✗ | ✗ |
| MAERI [31] | ✓ | ✗ | ✓ | ✗ | ✗ | ✗ | ✗ |
| MAESTRO [30] | ✓ | ✗ | ✓ | ✗ | ✗ | ✓ | ✗ |
| HAQ [49] | ✓ | ✗ | ✗ | ✓ | ✗ | ✗ | ✗ |
| Accelergy [50] | ✓ | ✗ | ✓ | ✓$^1$ | ✗ | ✗ | ✗ |
| Timeloop [36] | ✓ | ✗ | ✓ | ✓$^2$ | ✗ | ✓ | ✗ |
| Gemmini [11] | ✓ | ✓ | ✗ | ✓$^3$ | ✗ | ✗ | ✗ |
| QUIDAM (Ours) | ✓ | ✓ | ✓ | ✓$^4$ | ✓ | ✓ | ✓ |

1: Supports INT8, INT16, FP32
2: Supports INT8, INT16, FP32
3: Supports INT8, UINT32, FP32
4: Supports INT4, INT8, INT16, FP32

Similarly, SCALE-Sim [40] is a cycle accurate, systolic-array based DNN accelerator simulator. It has a python-based cycle accurate model that can generate results for hardware performance and utilization metrics. Although this tool can help rapid exploration of systolic-array based DNN accelerators for a given DNN layer, the built-in model is difficult to modify for a different hardware accelerator architecture and it lacks significant features such as quantization support, lightweight processing elements, and DNN accelerator and model co-exploration options.

As mapping deep neural networks onto DNN accelerators plays an important role in energy-efficiency, Eyeriss [2] analyzed various dataflow strategies in the existing DNN accelerator domain and came up with a novel approach called row-stationary dataflow which performs better than other dataflow strategies in terms of throughput and energy-efficiency [2]. However, Eyeriss is only an instance in the vast design space of DNN accelerators and its implementation is not open-source to foster future research on DNN accelerator and model co-exploration.

To this end, MAERI [31] and MAESTRO [30] have been proposed by the researchers to support spatial-array architectures and enrich the design space with various dataflows such as row-stationary dataflow which was originally proposed by Eyeriss [2]. MAERI [31] presents a new DNN accelerator architecture implemented with a set of modular and reconfigurable building blocks which can easily support various DNN mappings onto the accelerator by utilizing switches. On top of that, MAESTRO [30] presents an analytical cost model to predict the hardware cost of dataflows to perform a design space exploration. These open-source frameworks assist researchers to perform a design space exploration on various dataflows. However, they are incapable of supporting various bit precision levels and efficient processing elements that utilize cheap shift logic instead of expensive multipliers.

To further improve the computational efficiency of hardware accelerators, researchers have investigated tuning the optimal bit precision level for each layer of a deep neural network. To this end, HAQ [49] proposed optimizing the bit precision of DNN accelerators by proposing an automated hardware-aware quantization framework that leverages reinforcement learning to automatically tune the quantization scheme for a given hardware architecture by integrating the power and performance feedback signals coming from the hardware architecture in the design loop.

As HAQ reveals that the optimal bit precision levels on different hardware architectures and resource constraints are significantly different, recently Accelergy [50] and Timeloop [36] have been proposed to complement the design space exploration process by supporting different bit precision levels with an architecture-level energy estimation methodology [50] and a design space exploration framework [36] for DNN accelerators.

Accelergy [50] presents an architecture-level energy estimation methodology for hardware accelerators that takes in an architecture description and hardware activity statistics such as action counts which are based on a given workload that is needed to be generated by a separate performance model. Although Accelergy [50] proposes a fast energy estimation methodology for DNN accelerators, it is not capable of performing a design space exploration for hardware architectures by itself. To this end, Timeloop [36] presents a framework for evaluating and performing a design space exploration of DNN accelerators. Timeloop [36] proposes a generic template to describe DNN hardware accelerators and characterize deep learning workloads and provides a practical design space exploration tool to understand the trade-offs in designing DNN accelerators across different workloads.

Although these tools perform preliminary analysis on the design space for DNN accelerators in different aspects, they do not incorporate specialized quantization-aware lightweight processing elements and they do not generate or share a highly-parameterized RTL implementation of the chosen design based on the input hardware configuration which is a significant impediment for enabling deployment of DNNs onto edge devices, as the actual deployment of the hardware design takes a significant amount of engineering effort.

Finally, Gemmini [11] has been proposed as an open-source co-processor/accelerator generator framework that leverages a flexible architectural template to represent different accelerator architectures. Moreover, Gemmini [11] provides a parameterized RTL implementation to enable hardware architects to gain more control onto the design and potentially gain subtle insights on how different architectural decisions affect overall performance of the system. To enable system researchers to investigate and run software stacks on the generated hardware accelerator, Gemmini [11] also supports a full system-on-chip environment. From the hardware implementation point of view, the internal DNN accelerator implementation is a systolic-array based architecture similar to SCALE-Sim [40] which is a relatively more simplistic architecture when compared to spatial-array based dataflow architectures [2]. Therefore, it does not support more sophisticated dataflows such as row-stationary dataflow and quantization-aware lightweight processing elements which can enlarge the hardware accelerator design space and push the Pareto-frontier even further in terms of accuracy and hardware efficiency metrics such as energy-efficiency and performance per area. Moreover, this framework does not have a pre-characterized analytical hardware cost model. Therefore, it is not suitable for a rapid design space exploration of DNN accelerators and DNN accelerator and model co-exploration research. Therefore, there is a need for a framework that can assist system architects and machine learning researchers to quickly iterate over various hardware accelerator designs and DNN configurations while having an accurate hardware cost model and a flexible implementation to enable researchers to easily build their novel ideas on top of it without going through the tedious and laborious effort of RTL implementation from scratch.

To this end, we propose *QUIDAM*, a highly parameterized spatial-array based DNN accelerator framework that has implicit optimizations for CNNs as it is a domain-specific architecture but also has sufficient flexibility in terms of changing the microarchitectural features of the architecture, enriching the design space by providing lightweight processing element implementations, and crucial DNN model parameters such as bit precision to fully support DNN accelerator and model co-exploration.

As the systems community investigates novel hardware architectures and tools to enable deployment of efficient inference on edge, the ML community has focused on model compression techniques to achieve these objectives. Specifically, pruning [15] and quantization [52] have received great interest in the ML community to design models for edge devices. In quantization, prior work focuses on improving fixed-point quantization [9, 26]. Additionally, hardware-friendly quantization schemes have also been proposed [8, 33, 46]. In this work, we implement a specific quantization scheme chosen for its hardware-efficiency as it relies on reduced representations using a limited sum of power-of-two [8].

## 3 METHODOLOGY

In this section, we present the proposed *QUIDAM* framework, as shown in Figure 1. First, the proposed *QUIDAM* framework takes hardware accelerator parameters and deep neural network configurations as inputs to obtain power, performance, and area models for the next stages. Figure 2 shows the available hardware accelerator and DNN configuration parameters that can be chosen using the *QUIDAM* framework. We show the implementation details and architectural components of our *QUIDAM* framework, as depicted in Figure 2 (Section 3.1). Moreover, we also detail the lightweight processing elements (LightPE) that we implemented in our framework to provide a specialized processing element (PE) type for quantized DNN models (Section 3.2). After developing the power, performance, and area models for the *QUIDAM* framework for a wide range of accelerator and DNN configurations (Section 3.3), we perform a Pareto-optimality analysis for accuracy and crucial hardware efficiency metrics such as performance per area and energy (Section 4.3-4.4). Finally, we carry out a DNN accelerator and model co-exploration analysis by generating candidate hardware and DNN configurations and demonstrate the generalizability of the proposed *QUIDAM* framework by co-exploring the design space of both hardware and DNN configurations (Section 4.5). Therefore, the proposed framework provides power, performance, area, and RTL implementation using the developed power, performance, and area models and parameterized RTL implementation. Our proposed framework fosters future research on design space co-exploration of DNN accelerators and DNN configurations.

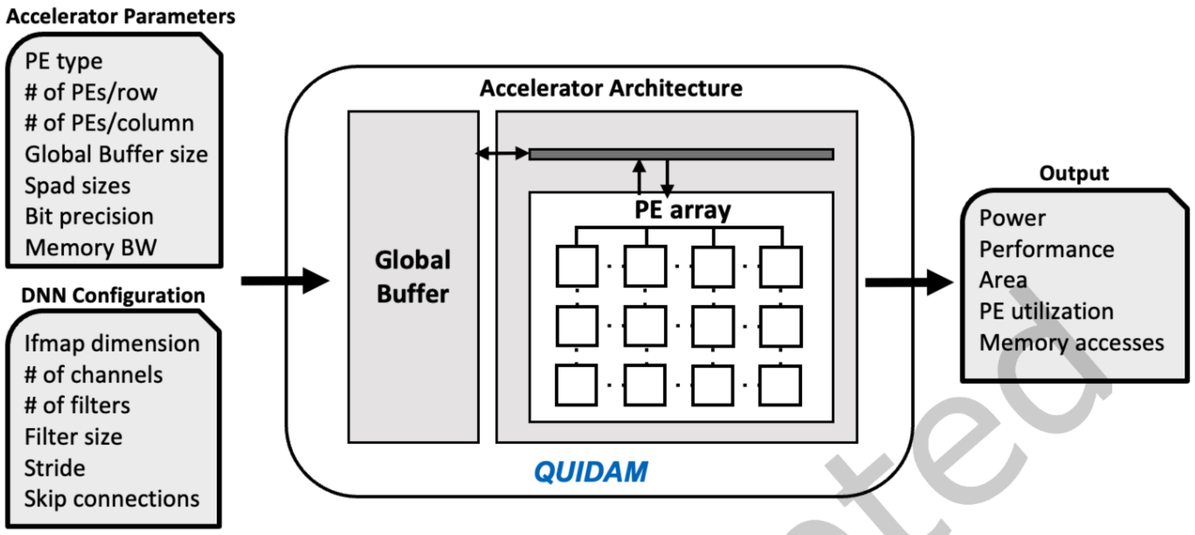


Fig. 2. Schematic depicting *QUIDAM* framework, with accelerator parameters and DNN configuration as inputs. The framework takes in accelerator parameters and layer-wise DNN configurations and generates power, performance, area results, and statistics on hardware utilization and memory accesses.

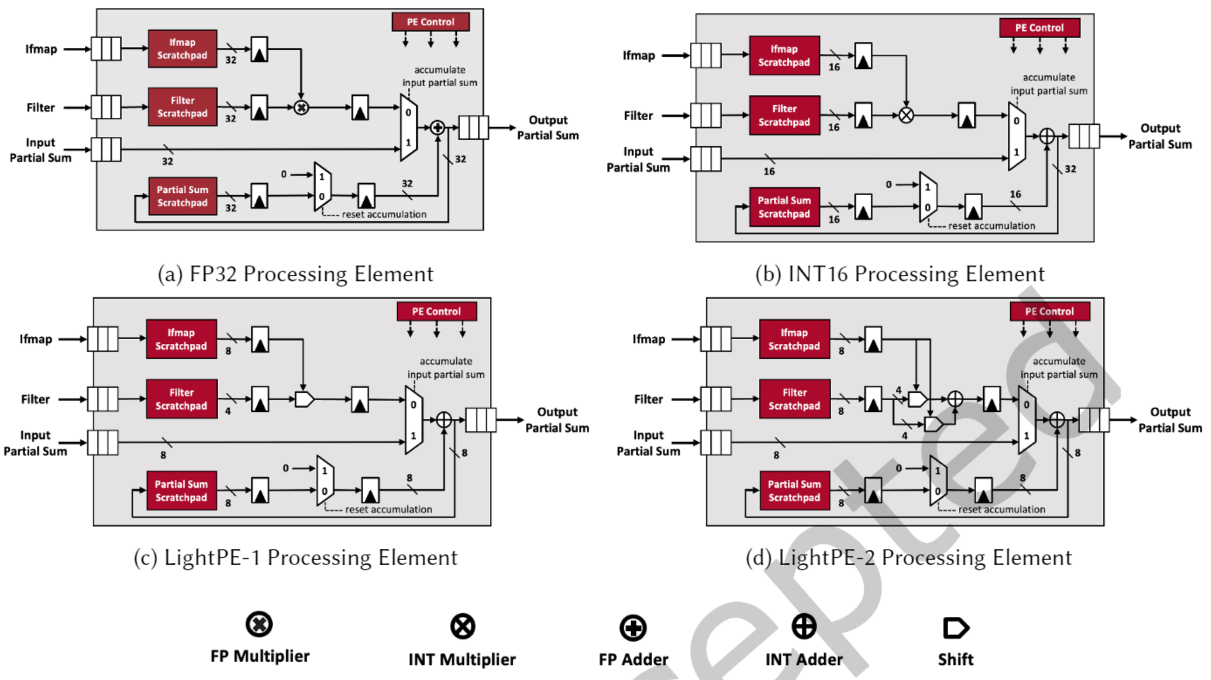
## 3.1 QUIDAM Framework

To enable comprehensive design space exploration for DNN accelerators for on-device machine learning, we implemented *QUIDAM*, a highly parameterized spatial-array based DNN accelerator framework in RTL. Our framework enables hardware designers and machine learning practitioners to rapidly iterate over various accelerator designs and DNN configurations and better understand trade-offs of different architectural components of the design for dizzying requirements of deploying machine learning models to edge devices. Moreover, hardware designers can also use the automatically generated RTL code to follow the design synthesis flow.

As depicted in Figure 2, *QUIDAM* framework is based on spatial-array based accelerators and utilizes row stationary dataflow which has been demonstrated to optimize the data movement in the storage hierarchy and improve the energy-efficiency of the system [2]. *QUIDAM* features a set of processing elements organized as a 2D array and a global buffer that stores input feature maps, filters, and activations. The number of PEs in each dimension can be tuned for different power, performance, and area requirements. Each PE includes an input feature map, a filter, partial sum scratchpads, and a multiply-accumulate (MAC) unit which can be implemented as a conventional MAC unit or a specialized shift-add unit based on the desired bit precision. Each of these architectural components can be tuned in a flexible and automated manner to perform a comprehensive design space co-exploration for on-device edge accelerators and DNN models. q

## 3.2 Lightweight Processing Elements (LightPE)

To enrich the design space of hardware accelerators and achieve a better Pareto-frontier for performance per area and energy-efficiency, we include LightPE implementations in our framework. LightPEs utilize 8 bits for activations, as well as 4 bits and 8 bits for weights for LightPE-1 and LightPE-2 designs, respectively. As 4 bit and 8 bit quantization techniques for on-device machine learning have become prevalent in various computing



(a) FP32 Processing Element

(b) INT16 Processing Element

(c) LightPE-1 Processing Element

(d) LightPE-2 Processing Element

Fig. 3. Detailed architectures of FP32, INT16, LightPE-1, and LightPE-2 processing elements.

platforms, we provide these specialized quantization-aware PE types in our *QUIDAM* framework to help hardware designers to enrich their design space and find better Pareto-frontiers.

Specifically, LightPEs use a special power-of-two quantization scheme [8] that quantizes the weights of the neural network into a limited sum of powers-of-two. In this case, the multiplication between the activation and weight can then be replaced by shifts and adds. More generally speaking, a multiplication between an 8-bit activation $x$ and an 8-bit weight $w$ can be formulated as follows:

$$\begin{aligned} y &= x \times w \\ &= \sum_{i=0}^{7} \mathbb{1}(w_i) \times (x << i) \\ \text{where } \mathbb{1}(w_i) &= \begin{cases} 1 & \text{the } i^{th} \text{ bit of } w \text{ is } 1 \\ 0 & \text{otherwise} \end{cases}, \end{aligned} \tag{1}$$

where $<<$ denotes a left shift operator. Based on equation 1, LightPE implementations approximate the sum with $k$ shifts and $k-1$ add operations, which was shown to be effective for significantly increasing energy-efficiency [7, 8]. We implement such an idea in LightPE-1 (for 1 shift) and LightPE-2 (for 2 shifts and one addition). In addition to LightPEs, we have also incorporated the design of a conventional 16 bit integer quantization (INT16) implementation and a conventional full-precision 32-bit floating point (FP32) implementation.

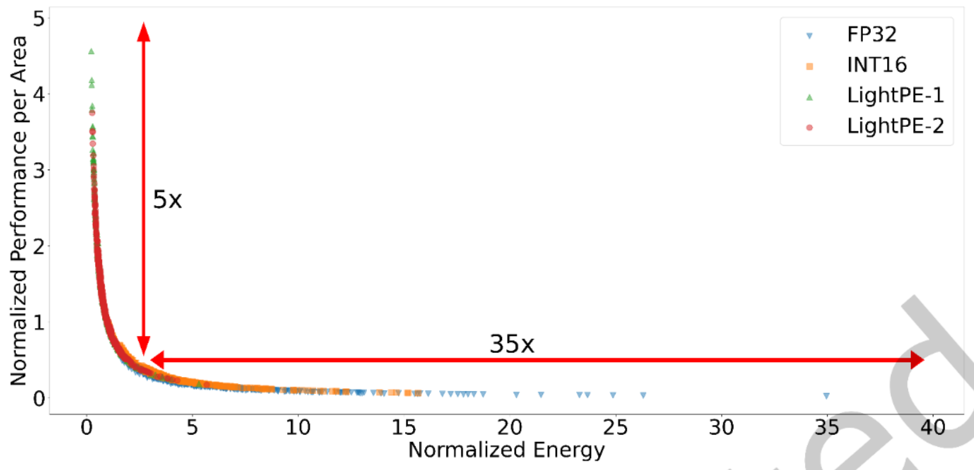


Fig. 4. Different PE types and bit precision lead to significant differences in performance per area and energy. Therefore, there is a need for a design space exploration framework that incorporates quantization-aware processing elements and rapidly iterate over various designs.

Figure 3 shows the detailed architectures of FP32, INT16, LightPE-1, and LightPE-2 processing elements. Each processing element has four FIFOs for input feature map (ifmap), filter, input partial sum, and output partial sum. Moreover, there are three scratchpad memories implemented in each processing element such as input feature map scratchpad, filter scratchpad, and partial sum scratchpad which get the data from aforementioned FIFOs. After getting the data from scratchpads, there are different versions of multiplication implementations between weights and activations for different processing elements. More specifically, Figure 3a shows the FP32 processing element that is used in *QUIDAM* framework. The FP32 implementation uses a 32-bit floating point multiplier and adder as it is commonly implemented in conventional systems. Figure 3b shows the INT16 processing element that has an 16-bit integer multiplier and adder. Figure 3c and Figure 3d show LightPE-1 and LightPE-2 implementations, respectively. The LightPE-1 implementation utilizes a shift instead of a multiplier and it uses 8 bits for activations and 4 bits for weights. To store a weight $w = \pm 2^{-m}$, where $m = 0, 1, ..., 7$, LightPE-1 needs four bits: one bit for the *sign(w)* and three bits for $|m|$. On the other hand, the LightPE-2 implementation utilizes two shifts and an addition as shown in Figure 3d. LightPE-2 utilizes 8 bits for activations and 8 bits for weights. To store a weight $w = \pm(2^{-m_1} + 2^{-m_2})$, where $m_1, m_2 = 0, 1, ..., 7$, LightPE-2 requires seven bits: one bit for the *sign(w)*, three bits for $|m_1|$, and three bits for $|m_2|$. For easier hardware implementation, 8 bits are used. After completing the multiplication and shift operations, all processing element implementations rely on a multiplexer for accumulating input partial sum data. Similarly, a second multiplexer is used for partial sum scratchpad to reset the accumulation. Finally, after the final addition operation, the data is sent to the output partial sum FIFO and the result is available.

Besides their low-precision benefits such as reducing the storage requirements, LightPEs also replace the multiplications with a more energy and area-efficient shift operation or a limited number of shifts and add operations [7, 8]. Therefore, LightPEs also achieve significant energy and area gains when compared to full-precision 32-bit floating point and 16-bit integer based designs. As a result, LightPEs provide an enriched design space for hardware designers and machine learning practitioners to analyze various trade-offs between accuracy and performance per area and energy.

To this end, we perform a design space exploration analysis with four different processing element types such as FP32, INT16, LightPE-1, and LightPE-2 and compare them in terms of normalized performance per area and normalized energy with respect to the INT16 design point with the highest performance per area. As seen in Figure 4, normalized energy varies 35× for almost the same performance per area region and normalized performance per area varies 5× for almost the same energy region. In addition, while most of the configurations are clustered around the knee of the scatter plot regardless of the quantization level, we note that FP32 configurations dominate the highest energy ones, while LightPE-1 configurations are the ones that push performance per area to highest values, orders of magnitude larger than the INT16 case. Therefore, different PE types and precision levels lead to significant differences in terms of performance per area and energy. These results also reinforce the need for a design space exploration framework that incorporates quantization-aware hardware and rapidly iterates over various designs.

## 3.3 Power, Performance, and Area Modeling

To build our quantization-aware power, performance, and area models, we use various hardware and DNN configurations. Specifically, to cover this comprehensive design space of hardware accelerators, we generate a variety of possible designs by varying global buffer size, number of PEs per row and column in the 2D PE array, bit precision, and PE type (FP32, INT16, LightPE-1, and LightPE-2). Within each PE, we also vary individual scratchpad sizes for input feature map, filter, and partial sum scratchpads.

We use Synopsys Design Compiler and the open-source FreePDK45 which is a commonly used process design kit [45] to synthesize our designs to obtain power, area, and initial timing results. We use Synopsys VCS RTL simulator to perform functional verification and collect timing information for various DNN configurations such as VGG-16 [44], ResNet-20, ResNet-34, ResNet-50, and ResNet-56 [16] that are implemented in our testbenches. After collecting power, area, and timing results from these tools, we use polynomial regression models and model selection techniques based on $k$-fold cross validation [35] to tune the degree of the polynomial.

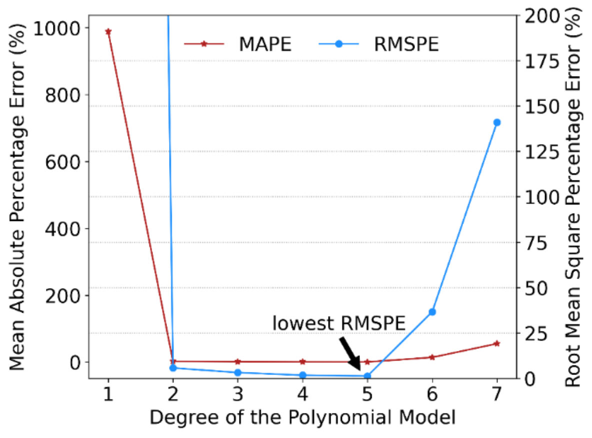
More concretely, a $K$-degree polynomial regression model is defined as:

$$F(\boldsymbol{x}) = \sum_j c_j \prod x_i^{q_{ij}}$$
$$\text{where } \boldsymbol{x} \in \mathbb{R}^d; q_{ij} \in \mathbb{N}; \forall j, \sum_i q_{ij} \leq K, \tag{2}$$

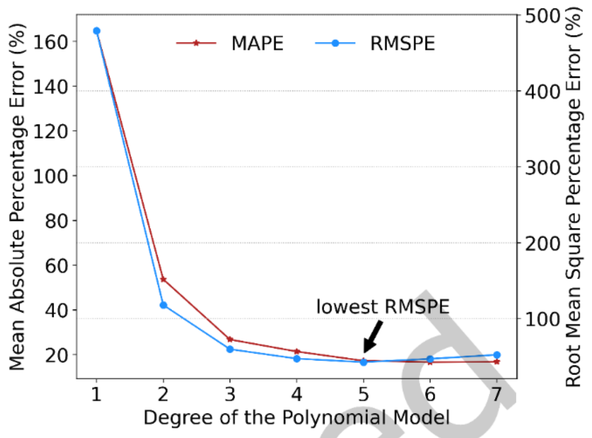
where $\boldsymbol{x}$ is a $d$-dimensional input feature vector with $x_i$ the $i^{\text{th}}$ feature, $j$ index denotes the $j^{\text{th}}$-term of the polynomial, and $\forall j,\ c_j$ are learnable coefficients. We note that using polynomial regression in modeling the power and performance characterizing deep neural networks when running on a fixed hardware (*i.e.*, desktop GPUs) has been studied before [1, 38]. To this end, for the novel design space where both the hardware and network configurations can vary, we explain in the sequel how features should be chosen for modeling. We detail the feature space for our proposed power, area, and latency models in the following.

**Power** We use Synopsys Design Compiler with inherently assumed switching activity when generating the power consumption. As a result, we choose $\boldsymbol{x}$ to be a four dimensional vector that includes: the scratchpad size for input feature map ($\text{SP}_{\text{if}}$), the scratchpad size for partial sum ($\text{SP}_{\text{ps}}$), the scratchpad size for filter weights ($\text{SP}_{\text{fw}}$), and the number of PEs (#PE). Finally, we develop individual models for each processing element type to improve the performance of our models as power depends on the PE type.

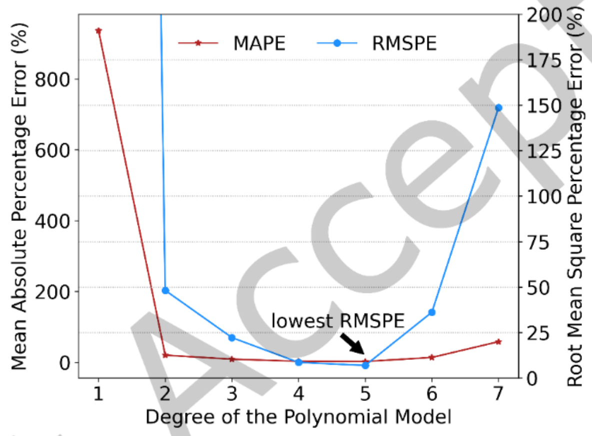
**Area** For area modeling, we use the same features as in power modeling because the features that affect power and area come from the same source. The area model only depends on the hardware configuration in contrast to the latency model which depends on both the hardware and the deep neural network configuration. More specifically, we choose $\boldsymbol{x}$ to be a four dimensional vector that includes: the scratchpad size for input feature



(a) Power Modeling



(b) Performance Modeling



(c) Area Modeling

Fig. 5. Comparison of the performance results of the model with respect to the degree of the polynomial model. A polynomial order of five is chosen for the power, performance, and area modeling since it achieves the lowest Root Mean Square Percentage Error (RMSPE) and Mean Absolute Percentage Error (MAPE) at the same time.

map ($SP_{if}$), the scratchpad size for partial sum ($SP_{ps}$), the scratchpad size for filter weights ($SP_{fw}$), and the number of PEs (#PE). Similar to power modeling, we build individual models for each processing element type as the arithmetic units differ between PE types.

**Latency** Latency depends on both the hardware and the neural network configurations. To cope with diverse network configurations, we adopt a layer-level latency modeling strategy. Specifically, we use the polynomial model to infer the per-layer latencies and sum them to obtain a network-level value. As a result, our training data for the polynomial model is at a layer-level granularity. We use a 12-dimensional feature vector for the latency model, which includes $SP_{if}$, $SP_{ps}$, $SP_{fw}$, the number of rows in the PE array ($PE_{rows}$), the number of columns in the PE array ($PE_{col}$), global buffer size (GBS), the input feature map dimension (A), the input channel count (C), filter count (F), kernel size (K), stride (S), and padding (P). For ResNets, we add two more binary features that

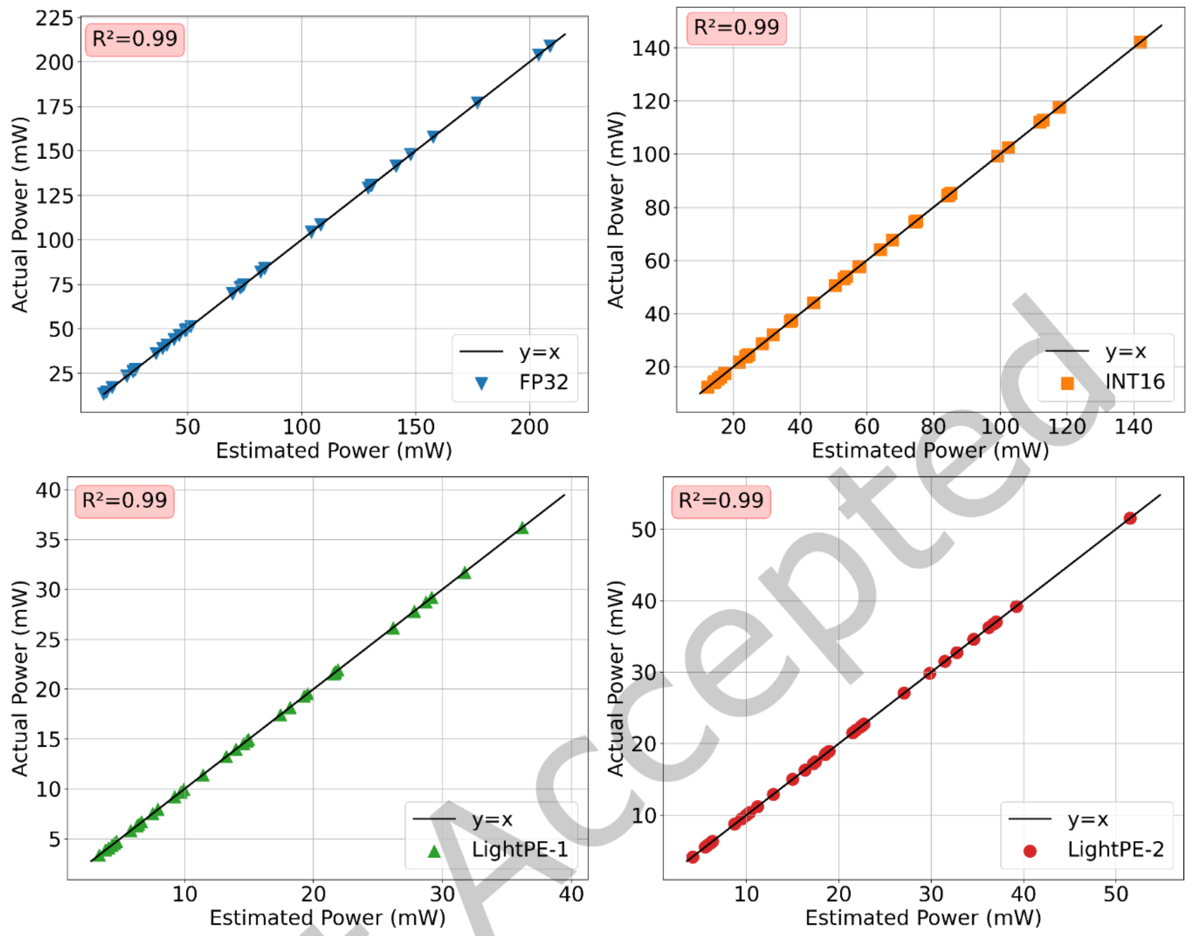


Fig. 6. Power estimation results for various processing element types such as FP32, INT16, LightPE-1, and LightPE-2. Each data point corresponds to a different hardware configuration that can be achieved by using the corresponding processing element type. As it can be seen, the proposed polynomial model agrees closely with the actual values extracted from the synthesis tools.

indicate whether the layer contains regular skip connection RS ∈ {0, 1} and dotted skip connection DS ∈ {0, 1}. Similar to power and area models, we build latency models specific to each processing element type to capture the correlation between PE implementations to latency. We use these latency models to obtain the performance results by taking the inverse of latency estimations. Therefore, we refer to the inverse of latency as performance throughout the paper.

Figure 5 depicts the model selection methodology used in our framework. We compare the mean absolute percentage error (MAPE) and root mean square percentage error (RMSPE) jointly to properly tune the model parameters of our polynomial model which we also apply model selection techniques based on *k*-fold cross validation [35] to tune the degree of the polynomial. As it can be seen, both MAPE and RMSPE results continuously decrease with a two degree of the polynomial model until a five degree of the polynomial model. Then, both

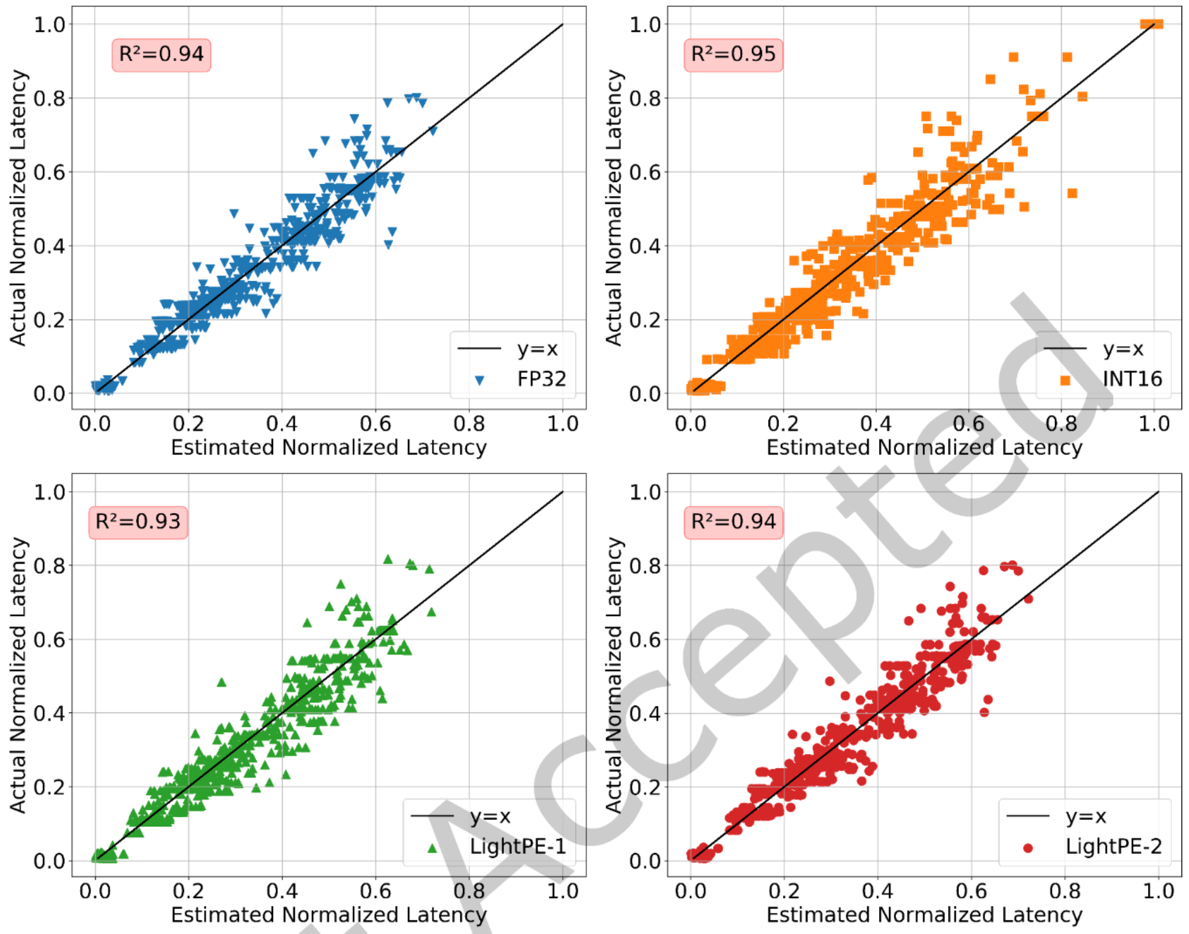


Fig. 7. Performance estimation results for various processing element types such as FP32, INT16, LightPE-1, and LightPE-2. Each data point corresponds to a different hardware configuration that can be achieved by using the corresponding processing element type. As it can be seen, the proposed polynomial model agrees closely with the actual values extracted from the synthesis tools.

MAPE and RMSPE results increase and get significantly higher. Therefore, for the power, performance, and area models, we use five degree polynomial models as both MAPE and RMSPE results are negligibly small as shown in Figure 5.

## 4 RESULTS

In this section, we present power, performance, and area modeling results for each processing element type and perform a design space exploration on various DNN models such as VGG-16 [44], ResNet-20, ResNet-34, ResNet-50, and ResNet-56 [16] on CIFAR-10, CIFAR-100, and ImageNet datasets to iterate through our framework to demonstrate the flexibility of *QUIDAM* for future studies.

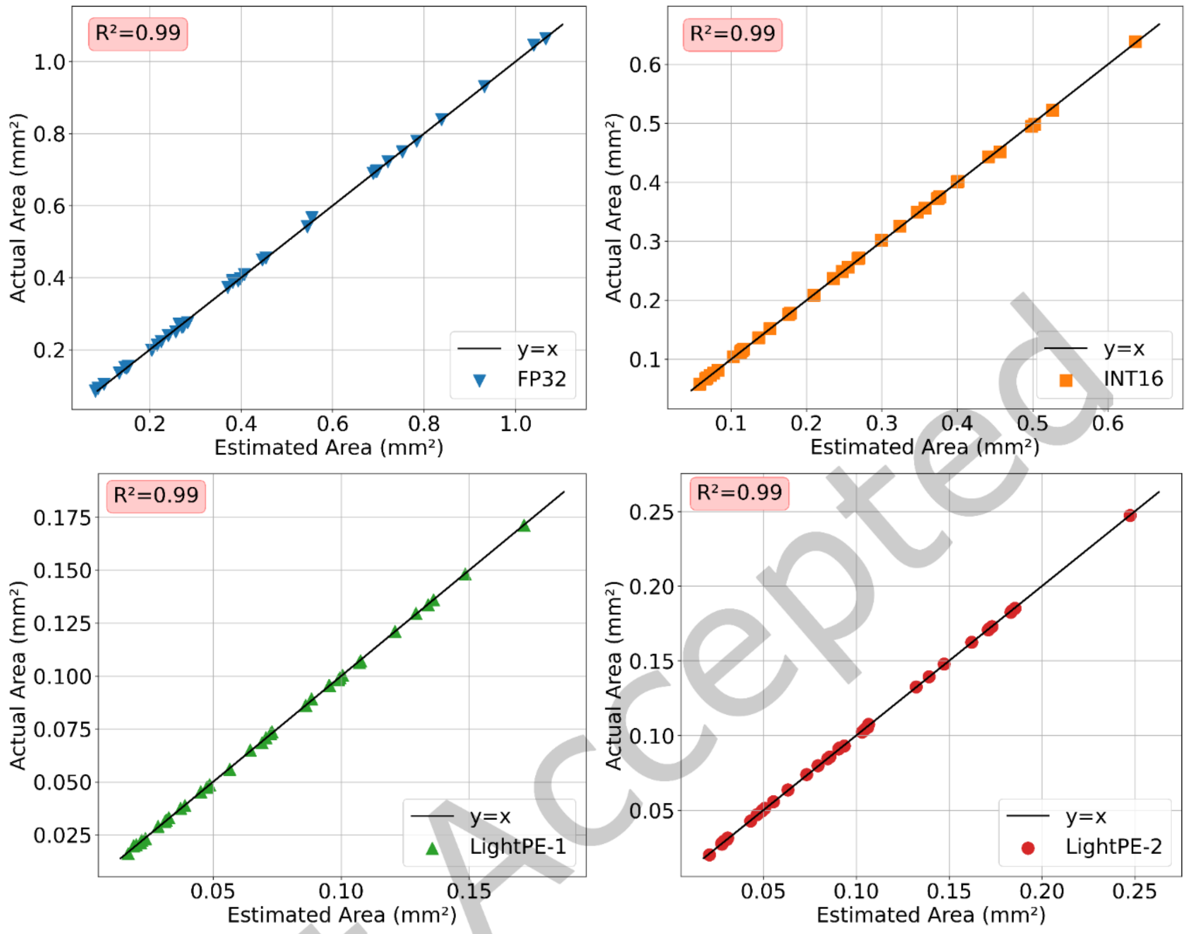


Fig. 8. Area estimation results for various processing element types such as FP32, INT16, LightPE-1, and LightPE-2. Each data point corresponds to a different hardware configuration that can be achieved by using the corresponding processing element type. As it can be seen, the proposed polynomial model agrees closely with the actual values extracted from the synthesis tools.

## 4.1 Power, Performance, and Area Model Accuracy

The power, performance, and area models detailed in Section 3 significantly speed up the design space exploration by 3-4 orders of magnitude. Indeed, *QUIDAM* enables fast design exploration since it reduces the characterization process from days for synthesizing the RTL implementation and determining power, performance, and area, results to seconds for using the trained models.

Figure 6-8 show the actual and estimated power, performance, and area results for each processing element type such as FP32, INT16, LightPE-1, and LightPE-2. Each data point in Figure 6-8 corresponds to a different hardware accelerator configuration in the comprehensive design space. As shown by the results, *QUIDAM*'s PPA models achieve high correlation to the actual PPA values.

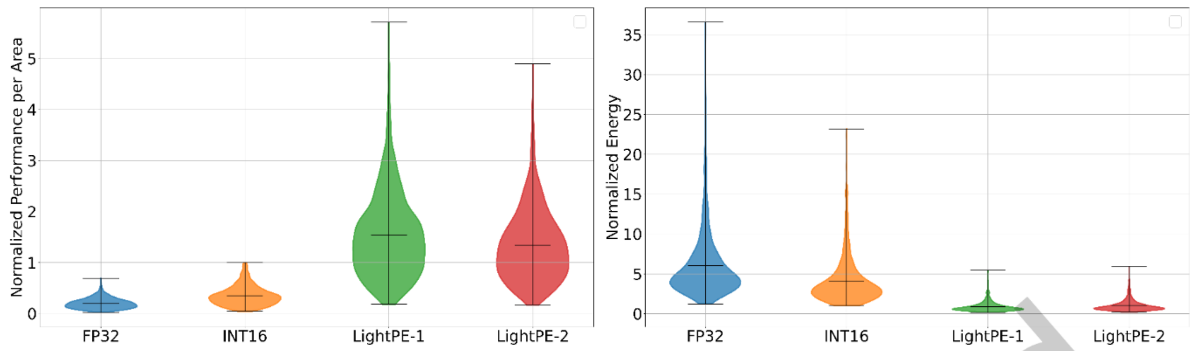


Fig. 9. Violin plots showing the full distribution of normalized performance per area (left chart) and normalized energy (right chart) results with respect to the best INT16 configuration for various processing element types such as FP32, INT16, LightPE-1, and LightPE-2. Each plot shows the full spectrum results for each PE type and black bars show the minimum, the maximum, and the median values. LightPEs provide higher performance per area and energy-efficient designs when compared to FP32 and INT16 based designs.

Figure 6,8 also show that the FP32 implementation has the highest area and power cost whereas LightPEs have the lowest area and power results when one processing element is considered. This shows the hardware-efficiency of LightPEs when compared to conventional PE implementations.

We also note that Figure 6 and Figure 8 show a higher correlation to actual power and area results than the corresponding chart for latency shown in Figure 7. This is expected because the performance results depend on both hardware accelerator and deep neural network configurations whereas the power and the area models have only hardware accelerator features. Therefore, building a close to perfect performance model is more difficult given the feature space dimensionality and richness.

## 4.2 Accelerator Design Space Exploration Results

To show the efficacy of LightPEs to conventional PE designs, we perform design space exploration on various DNN models such as VGG-16 [44], ResNet-20, ResNet-34, ResNet-50, and ResNet-56 [16] on CIFAR-10, CIFAR-100, and ImageNet datasets as shown in Figure 9. We show the normalized performance per area and normalized energy results for each PE type with respect to the baseline INT16 based implementation with the highest performance per area for the given design space. Performance per area is a useful metric as different processing element implementations use different bit precision and this affects the required storage of these implementations. Therefore, we use performance per area as a comparison metric in our analysis.

As it can be seen, LightPE implementations consistently outperform conventional INT16 and FP32 in both performance per area and energy objectives, which proves their efficacy in terms of hardware-efficiency. Figure 9 shows the full distribution including the minimum, the maximum, and the median results for each PE type. As it can be seen, LightPEs consistently provide higher performance per area and lower energy consuming design points as opposed to FP32 and INT16 based designs. Specifically, LightPE-1 and LightPE-2 achieve 4.8× and 4.1× more performance per area and 4.7× and 4× less energy on average across VGG-16, ResNet-20, and ResNet-56 workloads on CIFAR-10/CIFAR-100 and VGG-16, ResNet-34, and ResNet-50 workloads on ImageNet datasets when compared to the best INT16 hardware configuration, respectively. On the other hand, INT16 baseline implementation achieves 1.8× more performance per area and 1.5× less energy on average when compared to the best FP32 configuration.

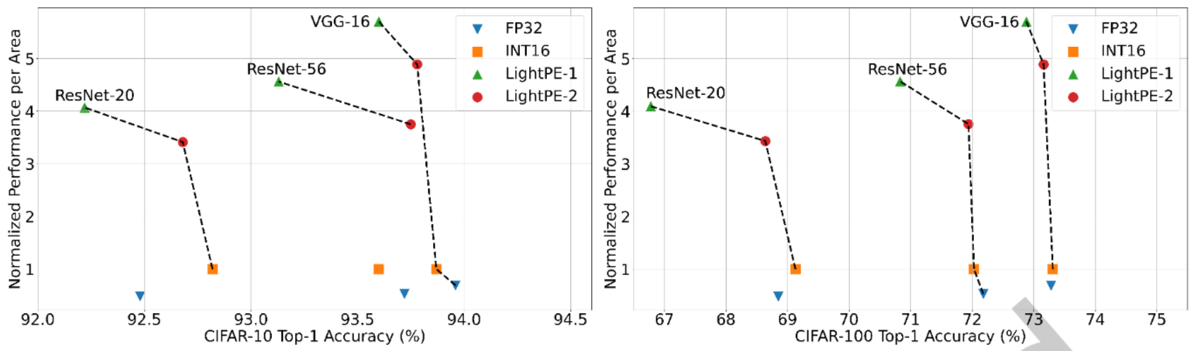


Fig. 10. Normalized performance per area and top-1 accuracy results for various processing element types such as FP32, INT16, LightPE-1, and LightPE-2 for CIFAR-10 (left chart) and CIFAR-100 (right chart). Each data point corresponds to the hardware configuration with the highest performance per area for the corresponding processing element type. Pareto-front is shown with a dashed line for each DNN model. LightPEs are consistently on Pareto-front for various DNN models.

These conclusions hold for all the models and the datasets considered in this work such as VGG-16, ResNet-20, ResNet-34, ResNet-50, and ResNet-56 thereby showing that the benefits of using lower precision generalize across a variety of models. We conclude that different bit precisions and PE types can lead to significantly different performance per area and energy results which are two critical metrics for machine learning and systems community strives to improve upon.

## 4.3 Pareto-Optimality for Accuracy and Performance per Area

To show the accuracy and performance per area trade-off for different processing element types, we perform a Pareto-front analysis by training VGG-16, ResNet-20, and ResNet-56 models for CIFAR-10 and CIFAR-100 datasets. For both datasets, we perform five runs for each DNN model and processing element type and plot the mean top-1 accuracy results. The training recipe for both CIFAR-10/CIFAR-100 datasets follows prior art [5, 17] which uses stochastic gradient descent with nesterov momentum, weight decay 0.0005, batch size 128, 0.1 initial learning rate with decrease by 5× at epochs 60, 120, and 160, and train for 200 epochs in total. We note that this training recipe is tuned for full-precision models. Therefore, the accuracy results for LightPE variants might be higher with proper hyperparameter tuning.

Figure 10 shows the normalized performance per area and accuracy results for FP32, INT16, LightPE-1, and LightPE-2. Performance per area results are normalized with respect to the best INT16 configuration for each DNN model. We plot the hardware configurations with the highest performance per area results for each processing element type. Next, we perform a Pareto-front analysis among different processing element types and show the Pareto-frontier with a dashed line for each DNN model. As it can be seen, LightPEs are consistently on the Pareto-front for various DNN models and datasets, whereas FP32 and INT16 based designs are occasionally dominated by LightPE variants mostly due to LightPE implementations pushing the Pareto-frontier by being more hardware-efficient in terms of performance per area and energy than FP32 and INT16 based designs. Moreover, in certain situations such as CIFAR-10 accuracy results LightPE-2 based design dominates FP32 and both INT16 and FP32 not only in hardware-efficiency metrics but also in accuracy results for ResNet-20 and ResNet-56 models, respectively. To sum up, LightPE-1 and LightPE-2 achieve on par accuracy results with FP32 and INT16 while achieving up to 5.7× and 4.9× more performance per area when compared to INT16 configuration, respectively.

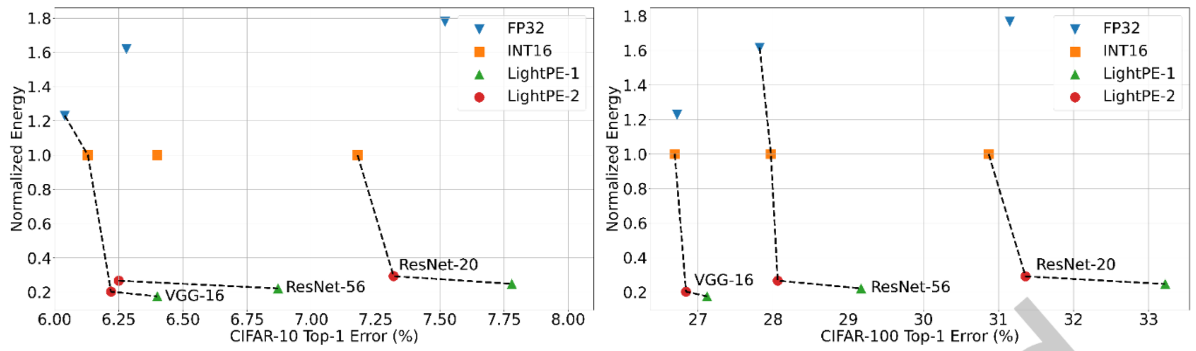


Fig. 11. Normalized energy and top-1 error results for various processing element types such as FP32, INT16, LightPE-1, and LightPE-2 for CIFAR-10 (left chart) and CIFAR-100 (right chart). Each data point corresponds to the hardware configuration with the lowest energy for the corresponding processing element type. Pareto-front is shown with a dashed line for each DNN model. LightPEs are consistently on Pareto-front for various DNN models.

## 4.4 Pareto-Optimality for Accuracy and Energy

We also perform a Pareto-front analysis for accuracy and energy results. We follow the same training methodology explained in Section 4-2. Figure 11 shows the normalized energy and accuracy results for FP32, INT16, LightPE-1, and LightPE-2 based designs. Energy results are normalized with respect to the best INT16 configuration for each DNN model. As it can be seen, LightPEs are systematically on Pareto-front for various DNN models and datasets. Specifically, LightPE-1 and LightPE-2 achieve 4.7× and 4× less energy on average across different workloads and datasets when compared to INT16 configuration, respectively. In addition, we note that as the model complexity increases, the accuracy gap between LightPEs and conventional FP32 and INT16 based designs decreases. Thus, we conclude that our proposed LightPEs have promising results for training larger models with negligible accuracy loss while achieving significant performance per area and energy improvements.

We summarize our findings in Table 2 which shows the Pareto-optimal results for each PE type for model accuracy and hardware-efficiency metrics such as performance per area and energy for VGG-16, ResNet-20, and ResNet-56 models on CIFAR-10 and CIFAR-100 datasets. Our results show that LightPE implementations provide on par accuracy results across models and datasets and improve the hardware-efficiency in terms of energy and performance per area when compared to FP32 and INT16 based designs. As it can be seen from Table 2, LightPEs consistently dominate FP32 and INT16 based designs in terms of hardware-efficiency metrics such as energy and performance per area. In addition, although we only claim that LightPEs can achieve similar accuracy to FP32 and INT16 designs, in certain situations such as ResNet-20 and ResNet-56 for CIFAR-10 dataset, the LightPE-2 based design achieves higher accuracy than the FP32 based design for ResNet-20, and both FP32 and INT16 based designs for ResNet-56, respectively.

Furthermore, Table 3 shows the clock frequency values found by *QUIDAM* for designs with different PE types. We note that LightPE based implementations provide up to 1.7× and 1.6× speedup when compared to FP32 and INT16 based designs, respectively. In addition, we note that LightPE-2 and LightPE-1 implementations achieve 435 MHz and 455 MHz in 45 nm technology node [45], respectively. As the Eyeriss design reports its core clock frequency as 200 MHz and it utilizes 65 nm technology node [4], we apply the prominent technology scaling rules to make a fair comparison among different designs. Based on these scaled calculations [41], we note that *QUIDAM* finds LightPE implementations that are 1.5× to 1.6× faster when compared to Eyeriss [4] design. Moreover, with the same INT16 based implementation, the *QUIDAM* generated DNN accelerator configuration achieves similar clock frequency (197 MHz) after technology scaling.

Table 2. Pareto-Optimal Results

| Network | PE Type | Accuracy (%) CIFAR-10 | Accuracy (%) CIFAR-100 | Energy | Performance per Area |
|---|---|---|---|---|---|
| VGG-16 | FP32 | **93.96** | 73.28 | 1.2× | 0.69× |
| | INT16 | 93.87 | **73.31** | 1× | 1× |
| | LightPE-2 | 93.78 | 73.16 | 0.20× | 4.9× |
| | LightPE-1 | 93.60 | 72.88 | **0.18**× | **5.7**× |
| ResNet-20 | FP32 | 92.48 | 68.85 | 1.8× | 0.48× |
| | INT16 | **92.82** | **69.13** | 1× | 1× |
| | LightPE-2 | 92.68 | 68.64 | 0.29× | 3.4× |
| | LightPE-1 | 92.22 | 66.78 | **0.25**× | **4.1**× |
| ResNet-56 | FP32 | 93.72 | **72.18** | 1.6× | 0.53× |
| | INT16 | 93.60 | 72.03 | 1× | 1× |
| | LightPE-2 | **93.75** | 71.94 | 0.27× | 3.8× |
| | LightPE-1 | 93.13 | 70.83 | **0.22**× | **4.6**× |

Table 3. Clock frequency values of *QUIDAM* generated designs with different PE types

| PE Type | Clock Frequency |
|---|---|
| FP32 | 275 MHz |
| INT16 | 285 MHz |
| LightPE-2 | 435 MHz |
| LightPE-1 | 455 MHz |

## 4.5 DNN Accelerator and Model Co-Exploration

So far, we have demonstrated the usefulness of our proposed framework by mainly varying the hardware architecture given some commonly adopted neural network designs. In this subsection, we demonstrate the generalizability of the proposed framework by co-exploring the design space of both hardware configurations and neural network architectures. To do so, we need an accuracy model for neural architectures in addition to the proposed power, performance, and area hardware cost models to rapidly iterate over different DNN models and perform a DNN accelerator and model co-exploration analysis. We note that an accuracy proxy model is needed since the search space of DNN architectures is extremely large (hundreds of thousands) which would require an untenable cost of training. To this end, we adopt the weight-sharing evaluation technique to estimate the accuracy of a candidate neural network architecture [12, 32]. More specifically, we first define a neural architecture search space for an existing VGG-16 network as shown in Table 4. The neural architecture search space used in our analysis is composed of Conv-BN-ReLU and MaxPool blocks with number of repetitions of each block ranging from 1 to 3, and number of channels ranging from 40 to 512 based on the layer number. It can be observed that the largest configuration is a VGG-16 architecture and there are smaller variants to be searched. The baseline VGG-16 model can be achieved by choosing the largest number of repetitions per block and the largest number of channels available in the search space shown in Table 4. The entire search space contains 110,592 candidate

Table 4. Search space for neural architectures used in DNN accelerator and model co-exploration analysis

| Block | Number of Repetitions | Channels |
|---|---|---|
| Conv-BN-ReLU | {1,2} | {40, 48, 56, 64} |
| MaxPool | 1 | N/A |
| Conv-BN-ReLU | {1,2} | {80, 96, 112, 128} |
| MaxPool | 1 | N/A |
| Conv-BN-ReLU | {1,2,3} | {160, 192, 224, 256} |
| MaxPool | 1 | N/A |
| Conv-BN-ReLU | {1,2,3} | {320, 384, 448, 512} |
| MaxPool | 1 | N/A |
| Conv-BN-ReLU | {1,2,3} | {320, 384, 448, 512} |
| MaxPool | 1 | N/A |

neural network architectures which is found by multiplying the number of all possible choices in each step of candidate architecture search. To obtain an accuracy predictor, we train the neural network by randomly sampling an architecture from the search space for each batch while sharing the weights with the largest neural network architecture [12, 32]. After the training is done, we randomly select 1, 000 network architectures and directly evaluate their accuracy on the validation set as our output for the accuracy predictor.

Figure 12 shows the normalized energy and the normalized area *vs.* top-1 model error results for various DNN accelerator and model pairs on the CIFAR-10 dataset. We randomly sample accelerator configurations and use 1000 DNN models. We then evaluate each accelerator and model pairs in terms of Pareto-optimality. Performance per area and energy results are normalized with respect to the minimum energy and area point in the INT16 design space, respectively. Figure 12 shows that LightPEs are consistently on the Pareto-front even when the DNN accelerator and model configurations are co-explored which shows the efficacy of LightPEs not only in a few commonly adopted DNN models but in a generalized DNN accelerator and model co-design space.

Based on these analyses and results, we conjecture that *QUIDAM* can successfully provide a wide range of DNN accelerator and model pairs based on different needs in terms of accuracy and critical hardware-efficiency metrics such as performance per area and energy. Therefore, we conclude that *QUIDAM* can be used for DNN accelerator and model co-design as it incorporates quantization-aware hardware and PPA models which significantly speeds up the co-design efforts.

## 5 CONCLUSION

In this work, we present *QUIDAM*, a quantization-aware highly parameterized DNN accelerator and model co-exploration framework. Our framework can foster future research on design space co-exploration of DNN accelerators for various design choices such as bit precision, processing element type, scratchpad size of processing elements, global buffer size, device bandwidth, number of total processing elements in the the design, and DNN configurations. Our results show that different bit precisions and processing element types lead to significant differences in terms of performance per area and energy. Specifically, LightPE-1 and LightPE-2 achieve 4.8× and 4.1× more performance per area and 4.7× and 4× energy improvement on average when compared to the best INT16 hardware configuration, respectively. We also show that our proposed LightPEs consistently achieve Pareto-optimal results in terms of accuracy and performance per area and energy for commonly used DNN models as well as when DNN accelerator and model configurations are co-explored. Therefore, design space

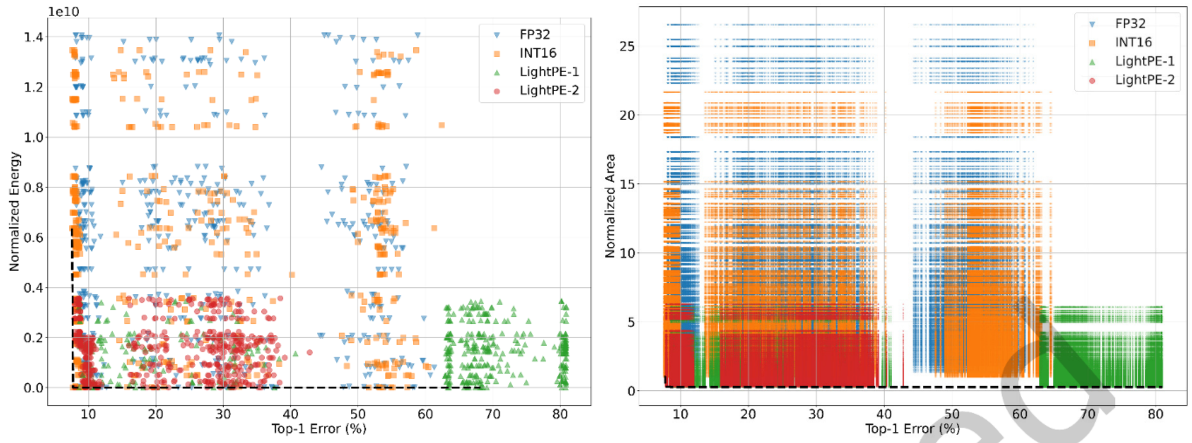


Fig. 12. Normalized energy (left chart) and normalized area (right chart) vs. top-1 error results for various DNN configurations and processing element types such as FP32, INT16, LightPE-1, and LightPE-2. Each data point corresponds to a different hardware and DNN architecture pair which are normalized to the minimum energy (left chart) and the minimum area (right chart) pair in the INT16 design space. Pareto-front for the co-exploration space is shown with a dashed line. As it can be seen, LightPEs are consistently on Pareto-front even when DNN accelerator and model configurations are co-explored.

co-exploration of quantization-aware DNN accelerators and models merits a meticulous analysis that takes these factors into account.

## ACKNOWLEDGMENTS

This research was supported in part by NSF CCF Grant No. 1815899 and NSF CSR Grant No. 1815780.

## REFERENCES

[1] Ermao Cai, Da-Cheng Juan, Dimitrios Stamoulis, and Diana Marculescu. 2017. Neuralpower: Predict and deploy energy-efficient convolutional neural networks. *Asian Conference on Machine Learning (ACML)* (2017), 622,637.

[2] Yu-Hsin Chen, Joel Emer, and Vivienne Sze. 2016. Eyeriss: A Spatial Architecture for Energy-efficient Dataflow for Convolutional Neural Networks. In *Proceedings of the 43rd International Symposium on Computer Architecture* (Seoul, Republic of Korea). IEEE Press, Piscataway, NJ, USA, 367–379. https://doi.org/10.1109/ISCA.2016.40

[3] Yu-Hsin Chen, Joel Emer, and Vivienne Sze. 2017. Using Dataflow to Optimize Energy Efficiency of Deep Neural Network Accelerators. *IEEE Micro* 37, 3 (2017), 12–21. https://doi.org/10.1109/MM.2017.54

[4] Yu-Hsin Chen, Tushar Krishna, Joel S. Emer, and Vivienne Sze. 52(1):127-138, 2017. Eyeriss: An Energy-Efficient Reconfigurable Accelerator for Deep Convolutional Neural Networks. *IEEE Journal of Solid-State Circuits* 52, 1 (Jan 52(1):127-138, 2017), 127–138. https://doi.org/10.1109/JSSC.2016.2616357

[5] Ting-Wu Chin, Ruizhou Ding, Cha Zhang, and Diana Marculescu. 2020. Towards Efficient Model Compression via Learned Global Ranking. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*.

[6] J. Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In *NAACL-HLT*.

[7] Ruizhou Ding, Zeye Liu, R. D. (Shawn) Blanton, and Diana Marculescu. 2018. Lightening the Load with Highly Accurate Storage- and Energy-Efficient LightNNs. *ACM Trans. Reconfigurable Technol. Syst.* 11, 3, Article 17 (Dec. 2018), 24 pages. https://doi.org/10.1145/3270689

[8] Ruizhou Ding, Zeye Liu, Rongye Shi, Diana Marculescu, and R.D. (Shawn) Blanton. 2017. LightNN: Filling the Gap between Conventional Deep Neural Networks and Binarized Networks. In *Proceedings of the on Great Lakes Symposium on VLSI 2017* (Banff, Alberta, Canada) *(GLSVLSI '17)*. Association for Computing Machinery, New York, NY, USA, 35–40. https://doi.org/10.1145/3060403.3060465

[9] Steven K. Esser, Jeffrey L. McKinstry, Deepika Bablani, Rathinakumar Appuswamy, and Dharmendra S. Modha. 2020. LEARNED STEP SIZE QUANTIZATION. In *International Conference on Learning Representations*. https://openreview.net/forum?id=rkgO66VKDS

[10] Mingyu Gao, Jing Pu, Xuan Yang, Mark Horowitz, and Christos Kozyrakis. 2017. TETRIS: Scalable and Efficient Neural Network Acceleration with 3D Memory. *SIGARCH Computer Architecture News* 45, 1 (2017), 751–764. https://doi.org/10.1145/3093337.3037702

[11] Hasan Genc, Seah Kim, Alon Amid, Ameer Haj-Ali, Vighnesh Iyer, Pranav Prakash, Jerry Zhao, Daniel Grubb, Harrison Liew, Howard Mao, Albert Ou, Colin Schmidt, Samuel Steffl, John Wright, Ion Stoica, Jonathan Ragan-Kelley, Krste Asanovic, Borivoje Nikolic, and Yakun Sophia Shao. 2021. Gemmini: Enabling Systematic Deep-Learning Architecture Evaluation via Full-Stack Integration. In *Proceedings of the 58th Annual Design Automation Conference (DAC)*.

[12] Zichao Guo, Xiangyu Zhang, Haoyuan Mu, Wen Heng, Zechun Liu, Yichen Wei, and Jian Sun. 2020. Single path one-shot neural architecture search with uniform sampling. In *European Conference on Computer Vision*. Springer, 544–560.

[13] Suyog Gupta and Berkin Akin. 2020. Accelerator-aware Neural Network Design using AutoML. *On-Device Intelligence Workshop in conjunction with MLSys* (2020).

[14] Song Han, Xingyu Liu, Huizi Mao, Jing Pu, Ardavan Pedram, Mark A Horowitz, and William J Dally. 2016. EIE: Efficient Inference Engine on Compressed Deep Neural Network. *International Conference on Computer Architecture (ISCA)* (2016).

[15] Song Han, Huizi Mao, and William J Dally. 2016. Deep Compression: Compressing Deep Neural Networks with Pruning, Trained Quantization and Huffman Coding. *International Conference on Learning Representations (ICLR)* (2016).

[16] K. He, X. Zhang, S. Ren, and J. Sun. 2016. Deep Residual Learning for Image Recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 770–778.

[17] Yang He, Guoliang Kang, Xuanyi Dong, Yanwei Fu, and Yi Yang. 2018. Soft Filter Pruning for Accelerating Deep Convolutional Neural Networks. In *Proceedings of the 27th International Joint Conference on Artificial Intelligence* (Stockholm, Sweden) *(IJCAI'18)*. AAAI Press, 2234–2240.

[18] Ahmet Inci, Evgeny Bolotin, Yaosheng Fu, Gal Dalal, Shie Mannor, David Nellans, and Diana Marculescu. 2020. The Architectural Implications of Distributed Reinforcement Learning on CPU-GPU Systems. *arXiv preprint arXiv:2012.04210* (2020).

[19] Ahmet Inci, Mehmet Meric Isgenc, and Diana Marculescu. 2021. Cross-Layer Design Space Exploration of NVM-based Caches for Deep Learning. *NVMW* (2021).

[20] Ahmet Inci, Mehmet Meric Isgenc, and Diana Marculescu. 2021. DeepNVM++: Cross-Layer Modeling and Optimization Framework of Non-Volatile Memories for Deep Learning. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* (2021), 1–1. https://doi.org/10.1109/TCAD.2021.3127148

[21] Ahmet Inci, Mehmet Meric Isgenc, and Diana Marculescu. 2022. Efficient Deep Learning Using Non-Volatile Memory Technology. *arXiv preprint arXiv:2206.13601* (2022).

[22] Ahmet Inci and Diana Marculescu. 2018. Solving the Non-Volatile Memory Conundrum for Deep Learning Workloads. *Architectures and Systems for Big Data Workshop in conjunction with ISCA* (2018).

[23] Ahmet Inci, Siri Garudanagiri Virupaksha, Aman Jain, Venkata Vivek Thallam, Ruizhou Ding, and Diana Marculescu. 2022. QADAM: Quantization-Aware DNN Accelerator Modeling for Pareto-Optimality. *arXiv preprint arXiv:2205.13045* (2022).

[24] Ahmet Inci, Siri Garudanagiri Virupaksha, Aman Jain, Venkata Vivek Thallam, Ruizhou Ding, and Diana Marculescu. 2022. QAPPA: Quantization-Aware Power, Performance, and Area Modeling of DNN Accelerators. *arXiv preprint arXiv:2205.08648* (2022).

[25] Ahmet Fatih Inci, Mehmet Meric Isgenc, and Diana Marculescu. 2020. DeepNVM: A Framework for Modeling and Analysis of Non-Volatile Memory Technologies for Deep Learning Applications. In *Proceedings of the 23rd Conference on Design, Automation and Test in Europe* (Grenoble, France) *(DATE '20)*. 1295–1298.

[26] Benoit Jacob, Skirmantas Kligys, Bo Chen, Menglong Zhu, Matthew Tang, Andrew Howard, Hartwig Adam, and Dmitry Kalenichenko. 2018. Quantization and training of neural networks for efficient integer-arithmetic-only inference. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2704–2713.

[27] Jun-Woo Jang, Sehwan Lee, Dongyoung Kim, Hyunsun Park, Ali Shafiee Ardestani, Yeongjae Choi, Channoh Kim, Yoojin Kim, Hyeongseok Yu, Hamzah Abdel-Aziz, Jun-Seok Park, Heonsoo Lee, Dongwoo Lee, Myeong Woo Kim, Hanwoong Jung, Heewoo Nam, Dongguen Lim, Seungwon Lee, Joon-Ho Song, Suknam Kwon, Joseph Hassoun, SukHwan Lim, and Changkyu Choi. 2021. Sparsity-Aware and Re-configurable NPU Architecture for Samsung Flagship Mobile SoC. In *2021 ACM/IEEE 48th Annual International Symposium on Computer Architecture (ISCA)*. 15–28. https://doi.org/10.1109/ISCA52012.2021.00011

[28] N. Jouppi, C. Young, Nishant Patil, David A. Patterson, Gaurav Agrawal, R. Bajwa, Sarah Bates, Suresh Bhatia, N. Boden, Al Borchers, Rick Boyle, Pierre luc Cantin, Clifford Chao, Chris Clark, Jeremy Coriell, Mike Daley, Matt Dau, J. Dean, Ben Gelb, T. Ghaemmaghami, R. Gottipati, William Gulland, R. Hagmann, C. R. Ho, Doug Hogberg, John Hu, R. Hundt, D. Hurt, J. Ibarz, A. Jaffey, Alek Jaworski, Alexander Kaplan, Harshit Khaitan, Daniel Killebrew, Andy Koch, Naveen Kumar, Steve Lacy, J. Laudon, James Law, Diemthu Le, Chris Leary, Zhuyuan Liu, Kyle A. Lucke, Alan Lundin, G. MacKean, A. Maggiore, Maire Mahony, K. Miller, R. Nagarajan, Ravi Narayanaswami, Ray Ni, K. Nix, Thomas Norrie, Mark Omernick, Narayana Penukonda, A. Phelps, Jonathan Ross, Matt Ross, Amir Salek, E. Samadiani, C. Severn, G. Sizikov, Matthew Snelham, J. Souter, D. Steinberg, Andy Swing, Mercedes Tan, G. Thorson, Bo Tian, H. Toma, Erick Tuttle, Vijay Vasudevan, Richard Walter, Walter Wang, Eric Wilcox, and D. Yoon. 2017. In-datacenter performance analysis of a tensor processing unit. *2017 ACM/IEEE 44th Annual International Symposium on Computer Architecture (ISCA)* (2017), 1–12.

[29] Norman P. Jouppi, Doe Hyun Yoon, Matthew Ashcraft, Mark Gottscho, Thomas B. Jablin, George Kurian, James Laudon, Sheng Li, Peter Ma, Xiaoyu Ma, Thomas Norrie, Nishant Patil, Sushma Prasad, Cliff Young, Zongwei Zhou, and David Patterson. 2021. Ten Lessons From Three Generations Shaped Google's TPUv4i : Industrial Product. In *2021 ACM/IEEE 48th Annual International Symposium on Computer Architecture (ISCA)*. 1–14. https://doi.org/10.1109/ISCA52012.2021.00010

[30] Hyoukjun Kwon, Prasanth Chatarasi, Michael Pellauer, Angshuman Parashar, Vivek Sarkar, and Tushar Krishna. 2019. Understanding Reuse, Performance, and Hardware Cost of DNN Dataflow: A Data-Centric Approach. In *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture, MICRO*. ACM, 754–768.

[31] Hyoukjun Kwon, Ananda Samajdar, and Tushar Krishna. 2018. MAERI: Enabling Flexible Dataflow Mapping over DNN Accelerators via Reconfigurable Interconnects. 461–475. https://doi.org/10.1145/3173162.3173176

[32] Liam Li and Ameet Talwalkar. 2020. Random search and reproducibility for neural architecture search. In *Uncertainty in Artificial Intelligence*. PMLR, 367–377.

[33] Yuhang Li, Xin Dong, and Wei Wang. 2020. Additive Powers-of-Two Quantization: An Efficient Non-uniform Discretization for Neural Networks. In *International Conference on Learning Representations*. https://openreview.net/forum?id=BkgXT24tDS

[34] Diana Marculescu, Dimitrios Stamoulis, and Ermao Cai. 2018. Hardware-Aware Machine Learning: Modeling and Optimization. In *2018 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)* (San Diego, CA, USA). IEEE Press, 1–8. https://doi.org/10.1145/3240765.3243479

[35] F. Mosteller and J. W. Tukey. 1968. Data analysis, including statistics. In *Handbook of Social Psychology, Vol. 2*, G. Lindzey and E. Aronson (Eds.). Addison-Wesley.

[36] Angshuman Parashar, Priyanka Raina, Yakun Sophia Shao, Yu-Hsin Chen, Victor A. Ying, Anurag Mukkara, Rangharajan Venkatesan, Brucek Khailany, Stephen W. Keckler, and Joel Emer. 2019. Timeloop: A Systematic Approach to DNN Accelerator Evaluation. In *2019 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*. 304–315. https://doi.org/10.1109/ISPASS.2019.00042

[37] Angshuman Parashar, Minsoo Rhu, Anurag Mukkara, Antonio Puglielli, R. Venkatesan, B. Khailany, J. Emer, Stephen W. Keckler, and W. Dally. 2017. SCNN: An accelerator for compressed-sparse convolutional neural networks. *ISCA* (2017).

[38] Hang Qi, Evan R. Sparks, and Ameet Talwalkar. 2017. Paleo: A Performance Model for Deep Neural Networks. In *Proceedings of the International Conference on Learning Representations*.

[39] Mohamed M. Sabry Aly, Mingyu Gao, Gage Hills, Chi-Shuen Lee, Greg Pitner, Max M. Shulaker, Tony F. Wu, Mehdi Asheghi, Jeff Bokor, Franz Franchetti, Kenneth E. Goodson, Christos Kozyrakis, Igor Markov, Kunle Olukotun, Larry Pileggi, Eric Pop, Jan Rabaey, Christopher Ré, H.-S. Philip Wong, and Subhasish Mitra. 2015. Energy-Efficient Abundant-Data Computing: The N3XT 1,000x. *Computer* 48, 12 (2015), 24–33. https://doi.org/10.1109/MC.2015.376

[40] Ananda Samajdar, Yuhao Zhu, Paul Whatmough, Matthew Mattina, and Tushar Krishna. 2018. SCALE-Sim: Systolic CNN Accelerator Simulator. *arXiv preprint arXiv:1811.02883* (2018).

[41] Satyabrata Sarangi and Bevan Baas. 2021. DeepScaleTool: A Tool for the Accurate Estimation of Technology Scaling in the Deep-Submicron Era. In *2021 IEEE International Symposium on Circuits and Systems (ISCAS)*. 1–5. https://doi.org/10.1109/ISCAS51556.2021.9401196

[42] Y. Shao, Jason Clemons, Rangharajan Venkatesan, B. Zimmer, Matthew R. Fojtik, Nan Jiang, Ben Keller, Alicia Klinefelter, N. Pinckney, Priyanka Raina, S. Tell, Yanqing Zhang, W. Dally, J. Emer, C. T. Gray, B. Khailany, and S. Keckler. 2019. Simba: Scaling Deep-Learning Inference with Multi-Chip-Module-Based Architecture. *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture* (2019).

[43] Yakun Sophia Shao, Brandon Reagen, Gu-Yeon Wei, and David Brooks. 2014. Aladdin: A pre-RTL, power-performance accelerator simulator enabling large design space exploration of customized architectures. In *2014 ACM/IEEE 41st International Symposium on Computer Architecture (ISCA)*. 97–108. https://doi.org/10.1109/ISCA.2014.6853196

[44] Karen Simonyan and Andrew Zisserman. 2015. Very Deep Convolutional Networks for Large-Scale Image Recognition. In *International Conference on Learning Representations*.

[45] J. E. Stine, I. Castellanos, M. Wood, J. Henson, F. Love, W. R. Davis, P. D. Franzon, M. Bucher, S. Basavarajaiah, J. Oh, and R. Jenkal. 2007. FreePDK: An Open-Source Variation-Aware Design Kit. In *MSE'07*.

[46] Thierry Tambe, En-Yu Yang, Zishen Wan, Yuntian Deng, Vijay Janapa Reddi, Alexander Rush, David Brooks, and Gu-Yeon Wei. 2020. Algorithm-hardware co-design of adaptive floating-point encodings for resilient deep learning inference. In *2020 57th ACM/IEEE Design Automation Conference (DAC)*. IEEE, 1–6.

[47] Mingxing Tan and Quoc Le. 2019. EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks, Kamalika Chaudhuri and Ruslan Salakhutdinov (Eds.), Vol. 97. PMLR, 6105–6114.

[48] Mingxing Tan, R. Pang, and Quoc V. Le. 2020. EfficientDet: Scalable and Efficient Object Detection. *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)* (2020), 10778–10787.

[49] Kuan Wang, Zhijian Liu, Yujun Lin, Ji Lin, and Song Han. 2019. HAQ: Hardware-Aware Automated Quantization With Mixed Precision. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

[50] Yannan N. Wu, Joel S. Emer, and Vivienne Sze. 2019. Accelergy: An Architecture-Level Energy Estimation Methodology for Accelerator Designs. In *IEEE/ACM International Conference On Computer Aided Design (ICCAD)*.
[51] Lei Yang, Zheyu Yan, Meng Li, Hyoukjun Kwon, Liangzhen Lai, Tushar Krishna, Vikas Chandra, Weiwen Jiang, and Yiyu Shi. 2020. Co-Exploration of Neural Architectures and Heterogeneous ASIC Accelerator Designs Targeting Multiple Tasks. *2020 57th ACM/IEEE Design Automation Conference (DAC)* (2020), 1–6.
[52] Shuchang Zhou, Yuxin Wu, Zekun Ni, Xinyu Zhou, He Wen, and Yuheng Zou. 2016. Dorefa-net: Training low bitwidth convolutional neural networks with low bitwidth gradients. *arXiv preprint arXiv:1606.06160* (2016).
[53] Yanqi Zhou, Xuanyi Dong, Berkin Akin, Mingxing Tan, Daiyi Peng, Tianjian Meng, Amir Yazdanbakhsh, Da Huang, Ravi Narayanaswami, and James Laudon. 2021. Rethinking Co-design of Neural Architectures and Hardware Accelerators. *arXiv preprint arXiv:2102.08619* (2021).