

Low-Power Neural Network Accelerators: Advancements in Custom Floating-Point Techniques

Yarib Nevarez

Universität Bremen

May 22, 2024

Introduction

- Advancements in AI/ML for IoT and TinyML

Introduction

- Advancements in AI/ML for IoT and TinyML
- Need for energy efficiency

Introduction

- Advancements in AI/ML for IoT and TinyML
- Need for energy efficiency
- Hybrid precision approach

Introduction

- Advancements in AI/ML for IoT and TinyML
- Need for energy efficiency
- Hybrid precision approach
- Design challenges

Introduction

- Advancements in AI/ML for IoT and TinyML
- Need for energy efficiency
- Hybrid precision approach
- Design challenges
- Approximate computing and quantization

Introduction

- Advancements in AI/ML for IoT and TinyML
- Need for energy efficiency
- Hybrid precision approach
- Design challenges
- Approximate computing and quantization
- Quality, interoperability, and compatibility

Goal and Objectives

- Goal: To develop advanced methodologies for energy-efficient neural network accelerators with custom floating-point (FP) computation in low-power and resource constrained devices.

Goal and Objectives

- Goal: To develop advanced methodologies for energy-efficient neural network accelerators with custom floating-point (FP) computation in low-power and resource constrained devices.
- Objectives:

Goal and Objectives

- Goal: To develop advanced methodologies for energy-efficient neural network accelerators with custom floating-point (FP) computation in low-power and resource constrained devices.
- Objectives:
 - Optimal custom FP number representation

Goal and Objectives

- Goal: To develop advanced methodologies for energy-efficient neural network accelerators with custom floating-point (FP) computation in low-power and resource constrained devices.
- Objectives:
 - Optimal custom FP number representation
 - Energy-efficient design strategies and custom FP arithmetic units

Goal and Objectives

- Goal: To develop advanced methodologies for energy-efficient neural network accelerators with custom floating-point (FP) computation in low-power and resource constrained devices.
- Objectives:
 - Optimal custom FP number representation
 - Energy-efficient design strategies and custom FP arithmetic units
 - Precision and quantization impact analysis

Goal and Objectives

- Goal: To develop advanced methodologies for energy-efficient neural network accelerators with custom floating-point (FP) computation in low-power and resource constrained devices.
- Objectives:
 - Optimal custom FP number representation
 - Energy-efficient design strategies and custom FP arithmetic units
 - Precision and quantization impact analysis
 - Hardware performance evaluation and bench marking

Goal and Objectives

- Goal: To develop advanced methodologies for energy-efficient neural network accelerators with custom floating-point (FP) computation in low-power and resource constrained devices.
- Objectives:
 - Optimal custom FP number representation
 - Energy-efficient design strategies and custom FP arithmetic units
 - Precision and quantization impact analysis
 - Hardware performance evaluation and bench marking
 - Practical application

Goal and Objectives

- Goal: To develop advanced methodologies for energy-efficient neural network accelerators with custom floating-point (FP) computation in low-power and resource constrained devices.
- Objectives:
 - Optimal custom FP number representation
 - Energy-efficient design strategies and custom FP arithmetic units
 - Precision and quantization impact analysis
 - Hardware performance evaluation and bench marking
 - Practical application
 - Future research

Outline

- 1 State-of-the-Art
- 2 Hybrid 8-bit Floating-Point and 4-bit Logarithmic Computation
- 3 Hybrid 6-bit Floating-Point Computation
- 4 Conclusions and Future Research

1 State-of-the-Art

2 Hybrid 8-bit Floating-Point and 4-bit Logarithmic Computation

3 Hybrid 6-bit Floating-Point Computation

4 Conclusions and Future Research

High-Performance FPGA-Based CNN Accelerator With Block-Floating-Point Arithmetic

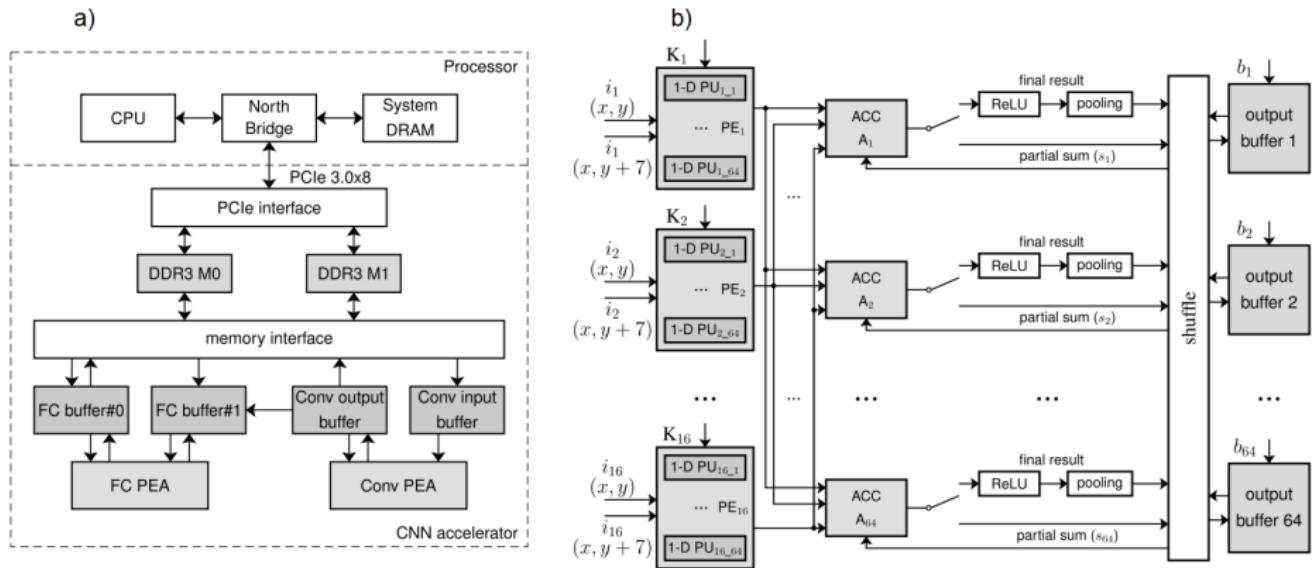


Figure: (a) System architecture. (b) Processing element array.

A 200MHz 202.4GFLOPS@10.8W VGG16 Accelerator in Xilinx VX690T

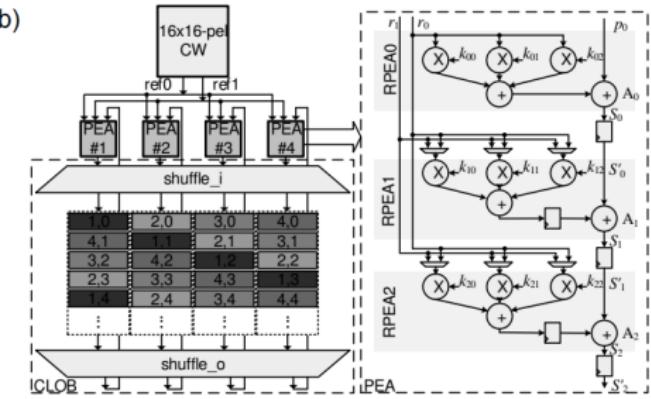
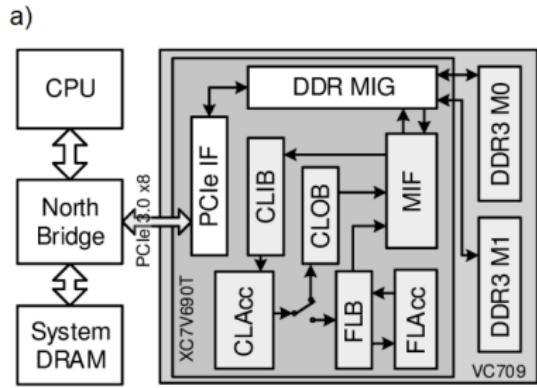


Figure: (a) System architecture. (b) Convolution accelerator.

Low-precision Floating-point Arithmetic for High-performance FPGA-based CNN Acceleration

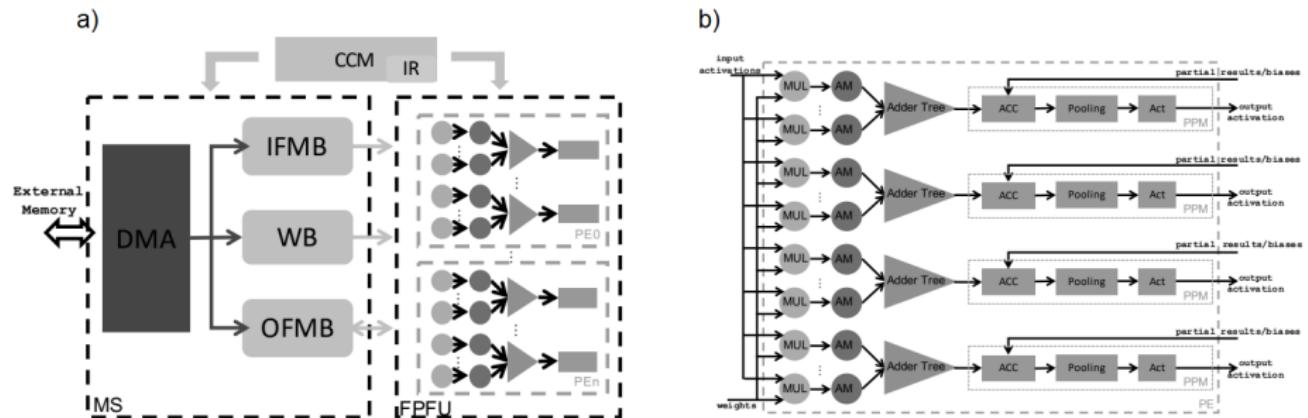


Figure: (a) System architecture. (b) Processing element.

CNN Hardware Acceleration on a Low-Power and Low-Cost APSoC

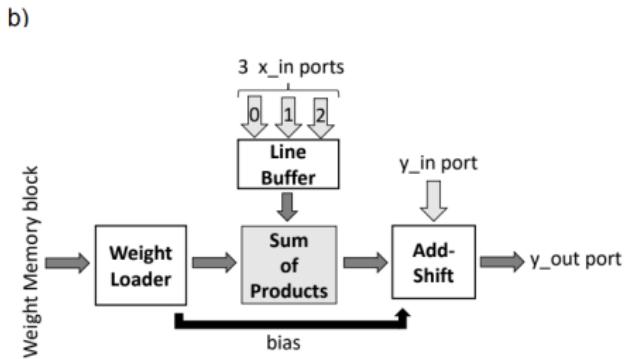
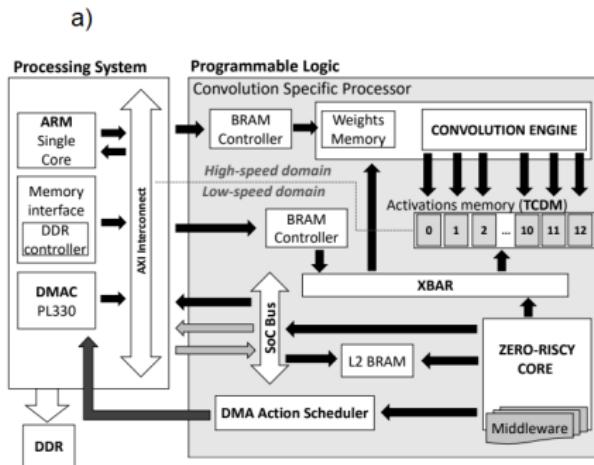


Figure: (a) System architecture. (b) Convolution engine.

1 State-of-the-Art

2 Hybrid 8-bit Floating-Point and 4-bit Logarithmic Computation

3 Hybrid 6-bit Floating-Point Computation

4 Conclusions and Future Research

Spike-by-Spike Neural Network

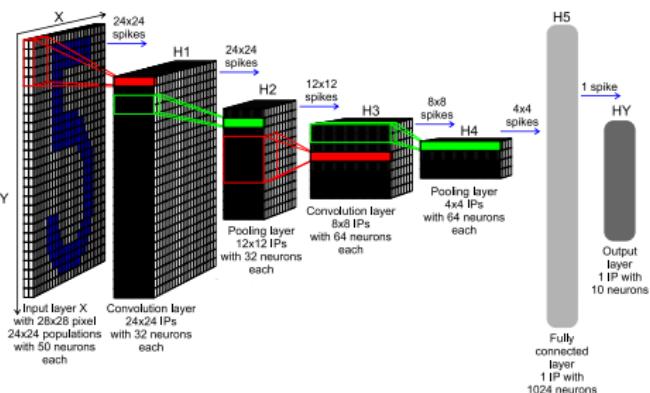


Figure: Spike-by-Spike (SbS) neural network architecture for handwritten digit classification task.

Spike-by-Spike Neural Network

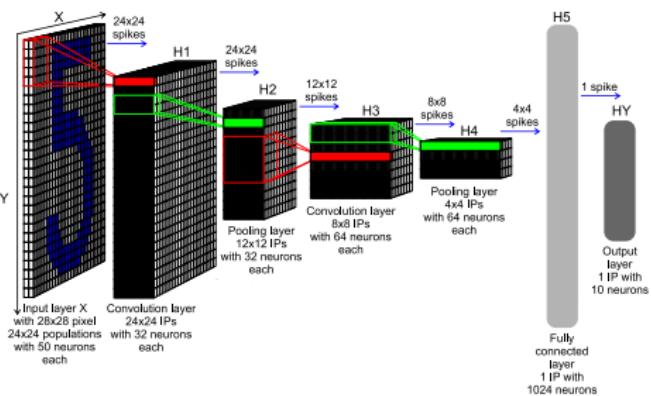


Figure: Spike-by-Spike (SbS) neural network architecture for handwritten digit classification task.

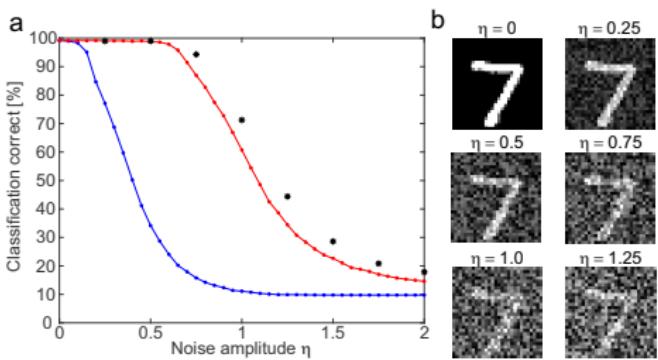


Figure: Performance classification of SbS NN versus equivalent CNN.

Spike-by-Spike Layer Update

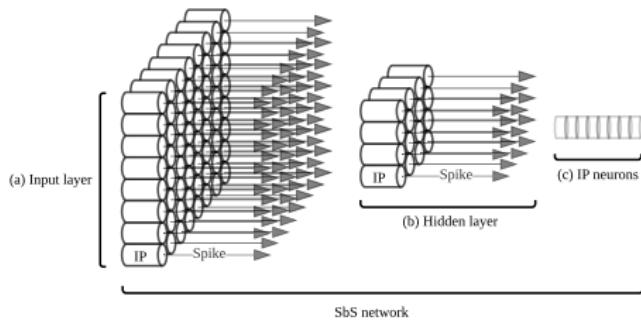


Figure: SbS inference population (IP) as independent computational entities.

$$h_{\mu}^{\text{new}}(i) = \frac{1}{1 + \epsilon} \left(h_{\mu}(i) + \epsilon \frac{h_{\mu}(i)W(s_t | i)}{\sum_j h_{\mu}(j)W(s_t | j)} \right)$$

HW/SW Co-Design Framework

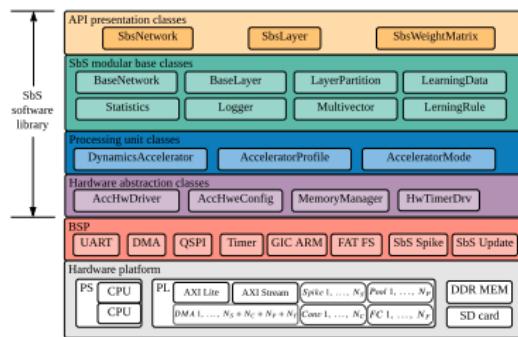


Figure: System-level overview of the embedded software architecture.

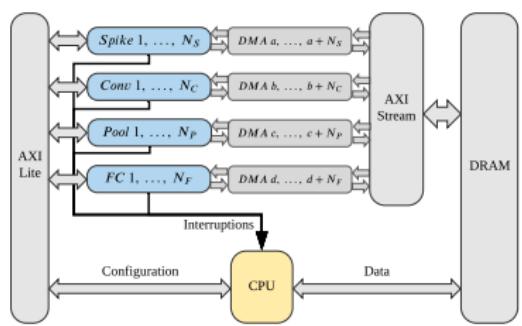


Figure: System-level hardware architecture with scalable number of heterogeneous processing units (PU).

Processing Unit

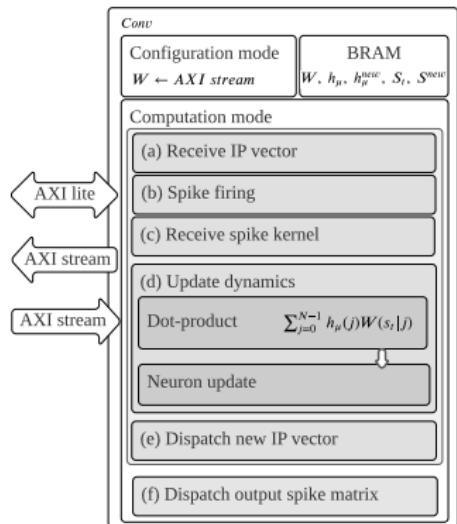


Figure: Conv processing unit.

Processing Unit

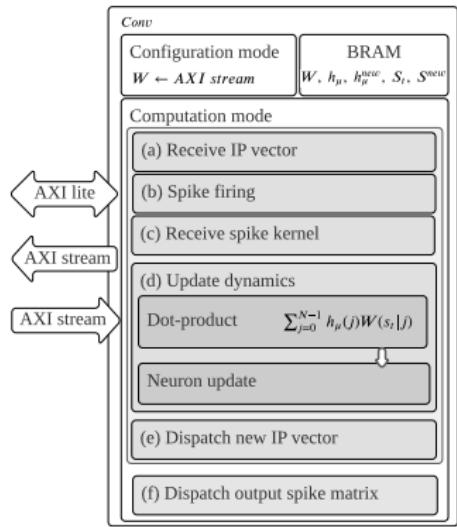


Figure: Conv processing unit.

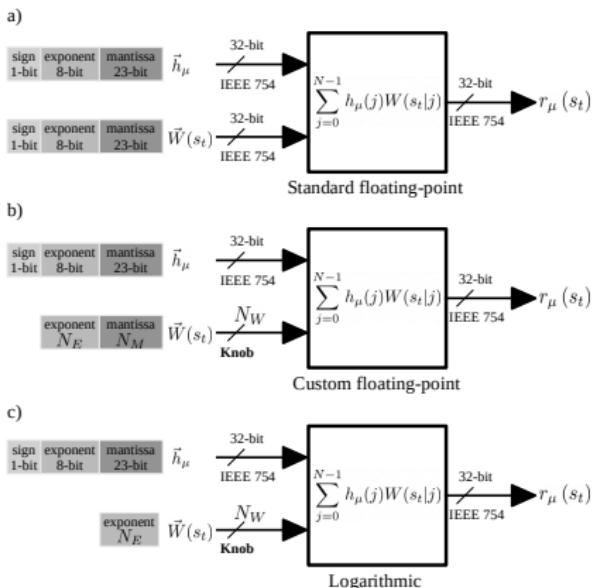


Figure: Dot-product hardware module.

Hybrid Dot-Product Approximation

$$r_\mu(s_t) = \sum_{j=0}^{N-1} h_\mu(j) W(s_t | j) \quad (1)$$

$$E_{\min} = \log_2(\min_i(W(i))) \quad (2)$$

$$N_E = \lceil \log_2(|E_{\min}|) \rceil \quad (3)$$

$$N_W = N_E + N_M \quad (4)$$

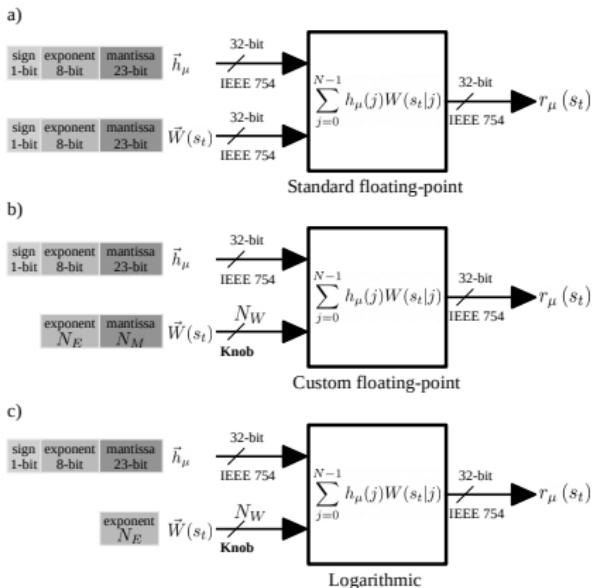


Figure: Dot-product hardware module.

Dot-Product with Standard Floating-Point (IEEE 754)

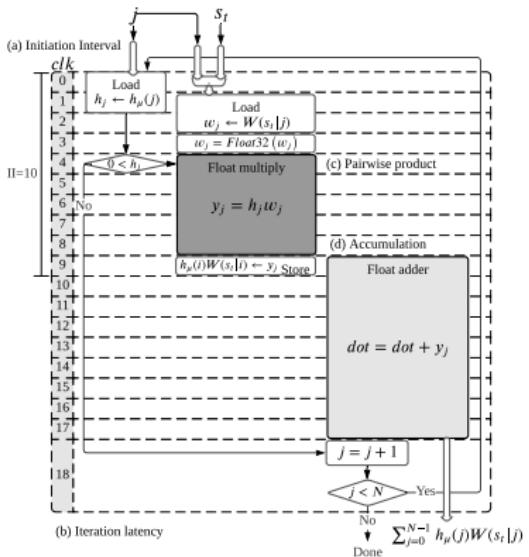


Figure: Dot-product hardware module with standard floating-point computation.

$$L_{f32} = 10N + 9$$

Dot-Product with Hybrid Custom Floating-Point Approximation

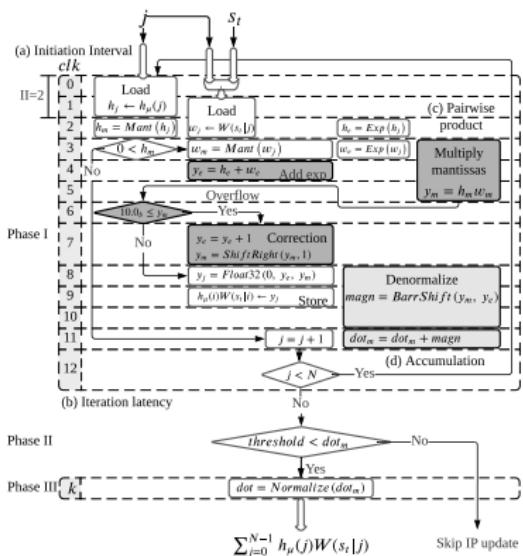


Figure: Dot-product hardware module with hybrid custom floating-point approximation.

$$L_{custom} = 2N + 11$$

Dot-product with Hybrid Logarithmic Approximation

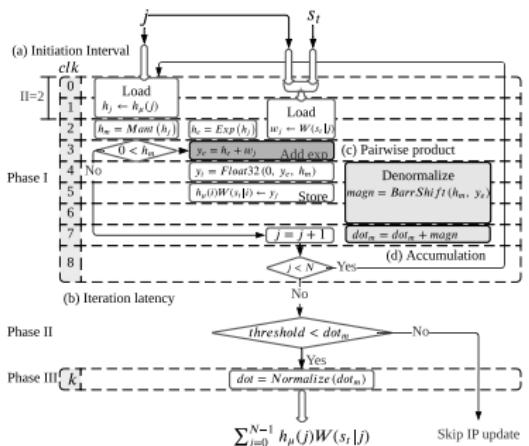


Figure: Dot-product hardware module with hybrid logarithmic approximation.

$$L_{custom} = 2N + 7$$

Acceleration with Standard Floating-Point

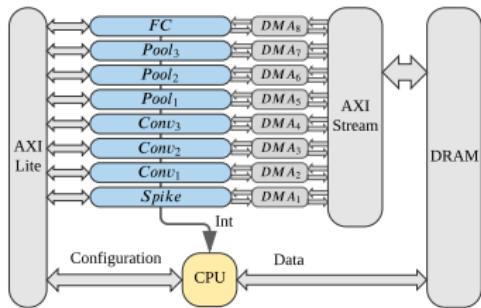


Figure: System overview of the top-level architecture with 8 processing units.

Acceleration with Standard Floating-Point

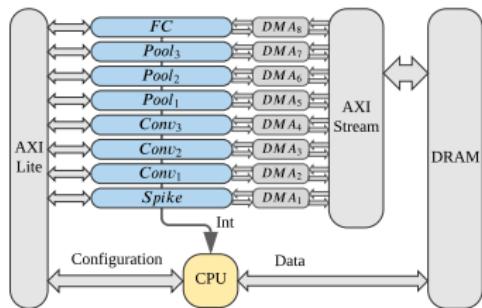


Figure: System overview of the top-level architecture with 8 processing units.

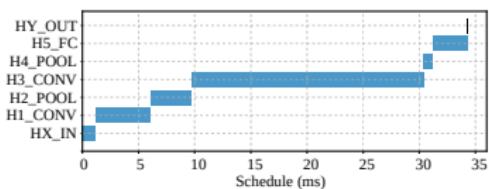


Figure: Computation on embedded CPU.

Acceleration with Standard Floating-Point

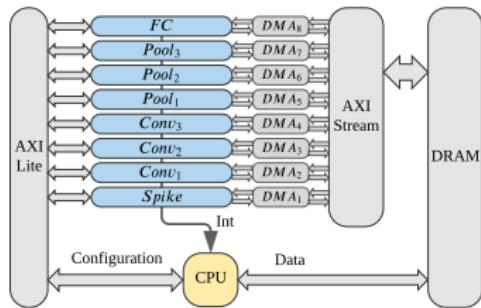


Figure: System overview of the top-level architecture with 8 processing units.

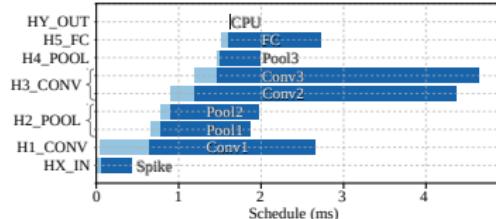


Figure: Performance of processing units with standard floating-point.

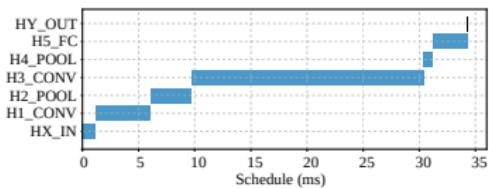


Figure: Computation on embedded CPU.

Acceleration with Standard Floating-Point

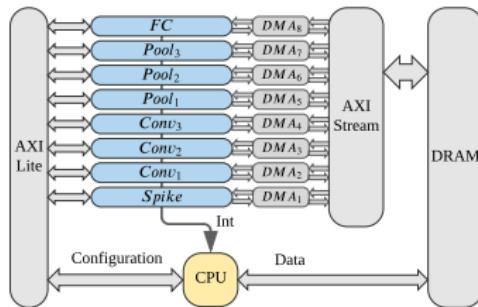


Figure: System overview of the top-level architecture with 8 processing units.

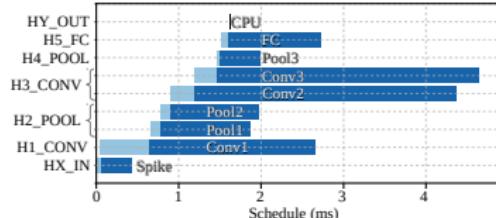


Figure: Performance of processing units with standard floating-point.

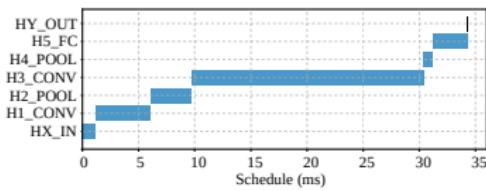


Figure: Computation on embedded CPU.

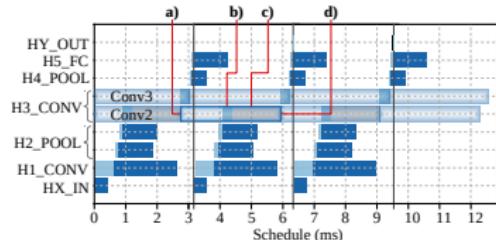


Figure: Performance bottleneck of cyclic computation on processing units with standard floating-point.

Acceleration with Custom Floating-Point

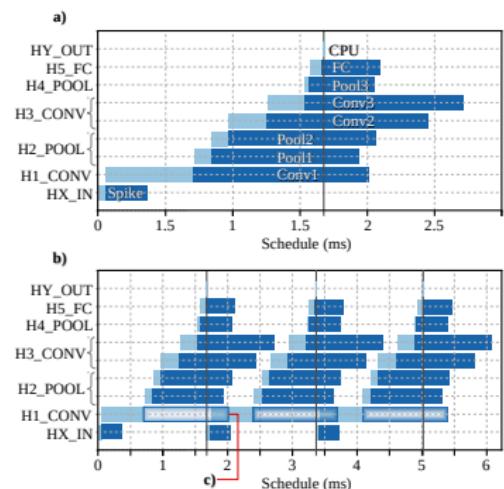


Figure: Performance on processing units with hybrid 8-bit floating-point.

Acceleration with Custom Floating-Point

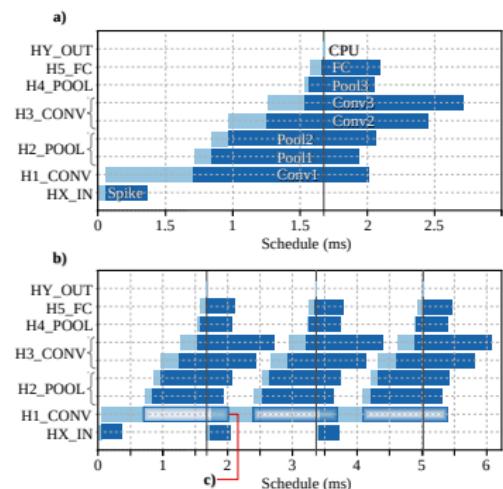


Figure: Performance on processing units with hybrid 8-bit floating-point.

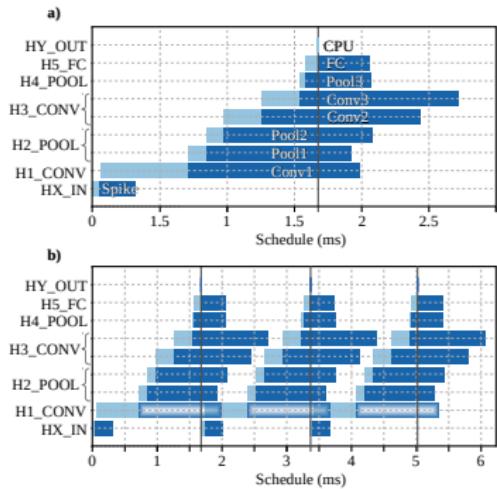


Figure: Performance of processing units with hybrid 4-bit logarithmic approximation.

Noise tolerance

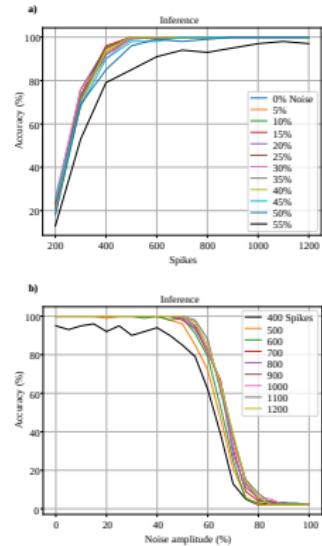


Figure: Noise tolerance with standard floating-point.

Noise tolerance

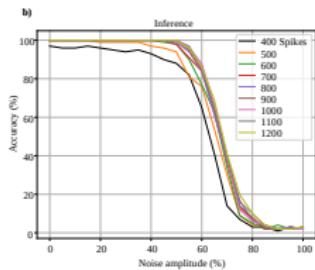
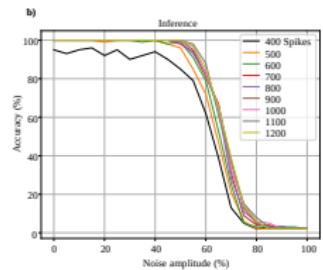
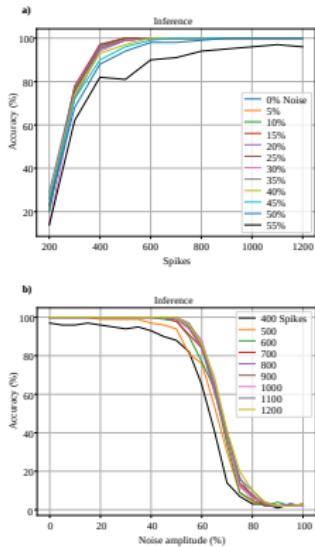
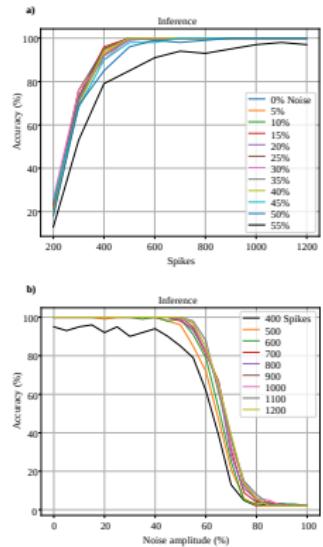


Figure: Noise tolerance with standard floating-point.

Figure: Noise tolerance with hybrid 8-bit floating-point approximation.

Noise tolerance

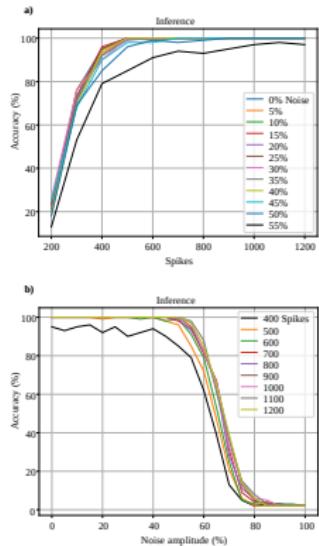


Figure: Noise tolerance with standard floating-point.

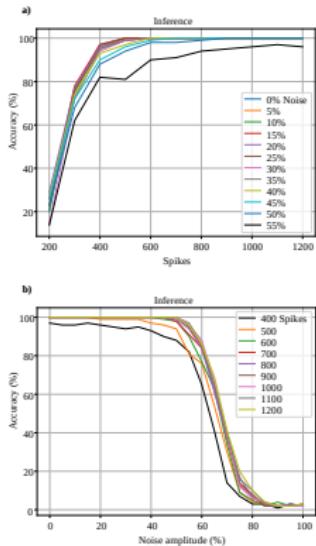


Figure: Noise tolerance with hybrid 8-bit floating-point approximation.

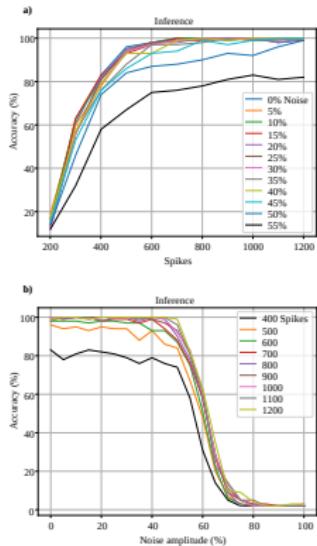


Figure: Noise tolerance with hybrid 4-bit logarithmic approximation.

Accelerator Implementations

Table: Accelerator implementations.

Platform implementation	Power (W)	Clk (MHz)	Latency (ms)	Acceleration	Accuracy (%)
Standard floating-point	2.420	200	3.18	10.7x	98.98
Hybrid floating-point 8-bit	2.369	200	1.67	20.5x	98.97
Hybrid Logarithmic 4-bit	2.324	200	1.67	20.5x	98.84

1 State-of-the-Art

2 Hybrid 8-bit Floating-Point and 4-bit Logarithmic Computation

3 Hybrid 6-bit Floating-Point Computation

4 Conclusions and Future Research

Convolution Operation

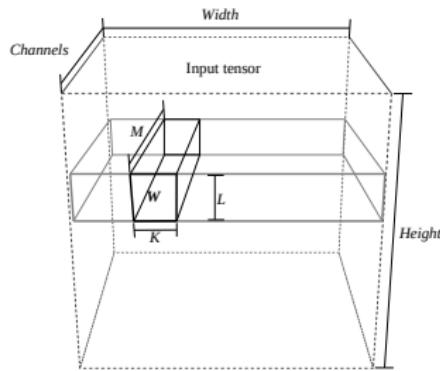


Figure: Two dimensional convolution operation.

$$\text{Conv2D}(W, b, h)_{i,j,o} = \sum_{k,l,m}^{K,L,M} h_{(i+k, j+l, m)} W_{(o, k, l, m)} + b_o$$

HW/SW Co-Design Framework

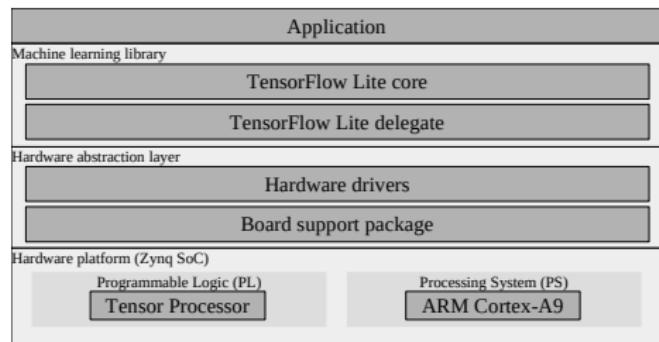


Figure: High level embedded software architecture.

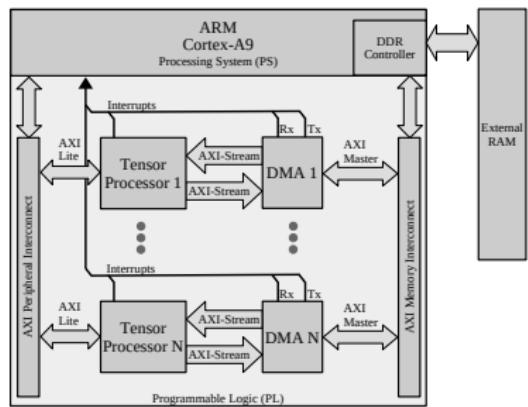


Figure: Base embedded system architecture.

Tensor Processor

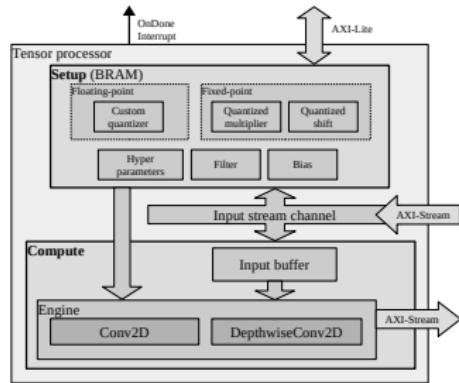


Figure: High level architecture.

Tensor Processor

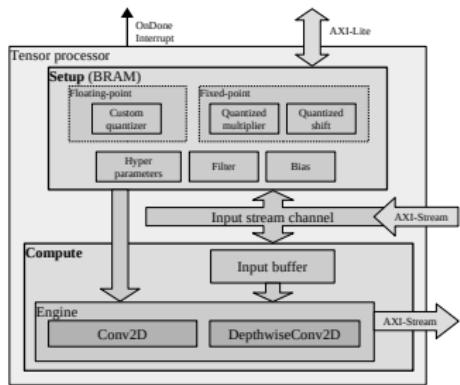


Figure: High level architecture.

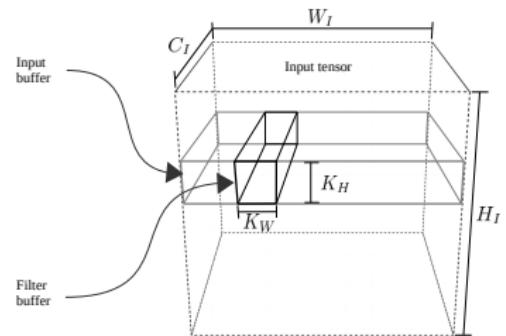


Figure: On-chip memory buffers.

On-Chip Memory

$$TP_M = TP_B + V_M \quad (5)$$

$$TP_B = Input_M + Filter_M + Bias_M \quad (6)$$

$$Input_M = C_I W_I C_I BitSize_I \quad (7)$$

$$Filter_M = C_I K_W K_H C_O BitSize_F \quad (8)$$

$$Bias_M = C_O BitSize_B \quad (9)$$

$$C_O = \frac{TP_M - V_M - K_H W_I C_I BitSize_I}{C_I K_W K_H BitSize_F + BitSize_B} \quad (10)$$

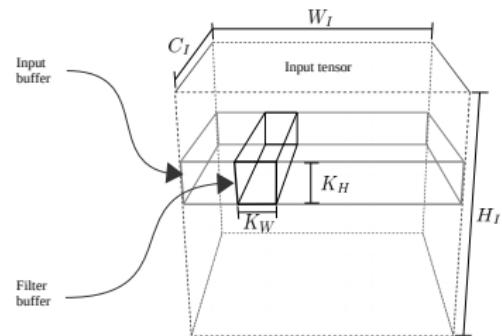


Figure: On-chip memory buffers.

Dot-Product with Hybrid Floating-Point 6-bit

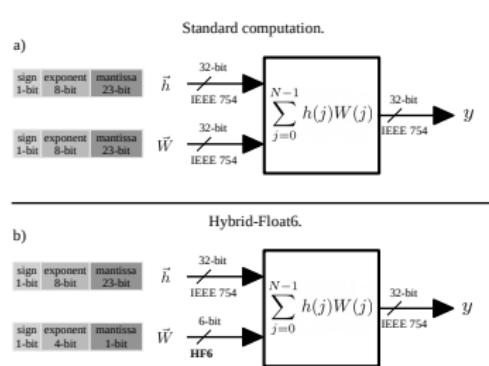
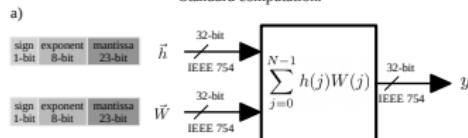


Figure: High level architecture.

Dot-Product with Hybrid Floating-Point 6-bit

a)

Standard computation.



b)

Hybrid-Float6.

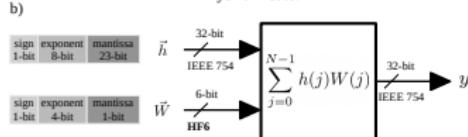


Figure: High level architecture.

Hybrid-Float6 MAC

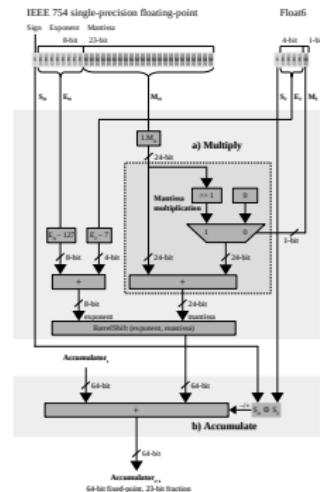


Figure: On-chip memory buffers.

$$L_{hf} = N + 7$$

Custom Floating-Point Quantization

Algorithm 1: Custom floating-point quantization.

Input: $MODEL$ as the CNN.
Input: E_{size} as the target exponent bit size.
Input: M_{size} as the target mantissa bits size.
Input: $STDM_{size}$ as the IEEE 754 mantissa bit size.
Output: $MODEL$ as the quantized CNN.

```

1 foreach layer in MODEL do
2   if layer is Conv2D or SeparableConv2D then
3     filter, bias ← GetWeights(layer)
4     foreach x in filter and bias do
5       sign ← GetSign(x)
6       exp ← GetExponent(x)
7       fullexp ←  $2^{E_{size}-1} - 1$  // Get full range value
8       cman ← GetCustomMantissa(x, Msize)
9       leftman ← GetLeftoverMantissa(x, Msize)
10      if exp < -fullexp then
11        | x ← 0
12      else
13        | if exp > fullexp then
14          | | x ← (-1)sign ·  $2^{fullexp} \cdot (1 + (1 - 2^{-M_{size}}))$ 
15        | | else
16          | | | if  $2^{STDM_{size}-M_{size}-1} - 1 < leftman$  then
17          | | | | cman ← cman + 1 // Above halfway
18          | | | | if  $2^{M_{size}} - 1 < cman$  then
19          | | | | | cman ← 0 // Correct mantissa overflow
20          | | | | | exp ← exp + 1
21          | | | | x ← (-1)sign ·  $2^{exp} \cdot (1 + cman \cdot 2^{-M_{size}})$ 
22      SetWeights(layer, filter, bias)

```

Acceleration in Sensor Analytics (TinyML)

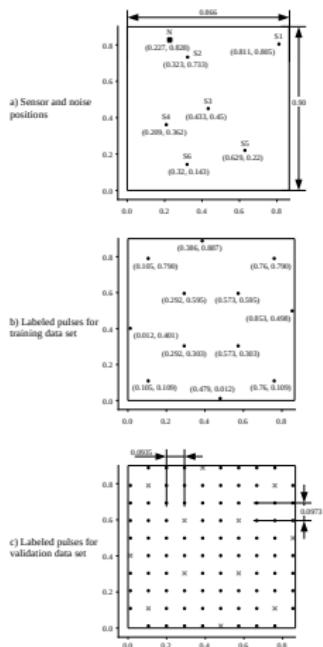


Figure: Structural health monitoring, all lengths are in metters (m).

Acceleration in Sensor Analytics (TinyML)

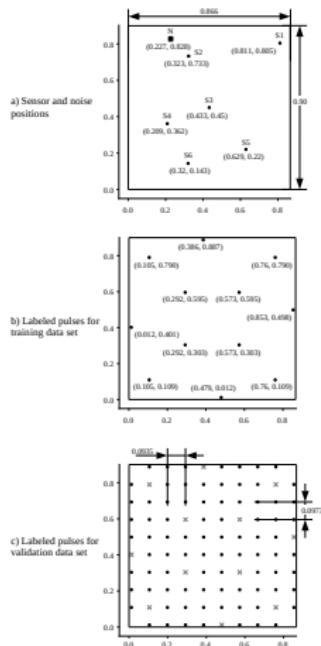
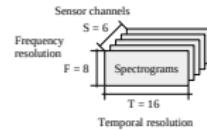


Figure: Structural health monitoring, all lengths are in metres (m).

a) Input tensor



CNN-regression model

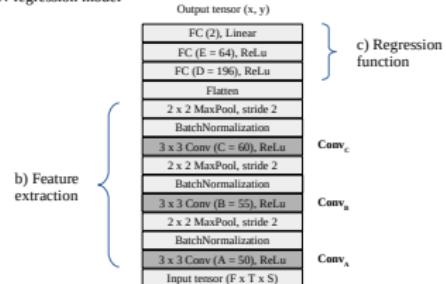


Figure: CNN-regression model for sensor analytics.

Training

Algorithm 2: Training with iterative early stop cycle.

```

Input: MODEL as the input model
Input:  $D_{train}$  as the training data set
Input:  $D_{val}$  as the validation data set
Input:  $N_I$  as the stop patience for iterative training cycle
Input:  $N_E$  as the early stop patience (epochs) for training
Input:  $B_{size}$  as the mini-batch size
Output: MODEL as the full-precision output model

// Initial training and evaluation
1 Train(MODEL,  $D_{train}, D_{val}, N_E, B_{size}$ )
2  $mse_i \leftarrow Evaluate(MODEL, D_{val})$ 
3  $n_I \leftarrow 0$ 
4 while  $n_I < N_I$  do
    // Iterative early stop cycle
    5   Train(MODEL,  $D_{train}, D_{val}, N_E, B_{size}$ )
    6    $mse_v \leftarrow Evaluate(MODEL, D_{val})$ 
    7   if  $mse_v < mse_i$  then
    8       |   Update(MODEL)
    9       |    $mse_i \leftarrow mse_v$ 
   10   end
   11   else
   12       |   MODEL  $\leftarrow LoadPreviousWeights()$ 
   13       |    $n_I \leftarrow n_I + 1$ 
   14   end
15 end

```

Training

Algorithm 2: Training with iterative early stop cycle.

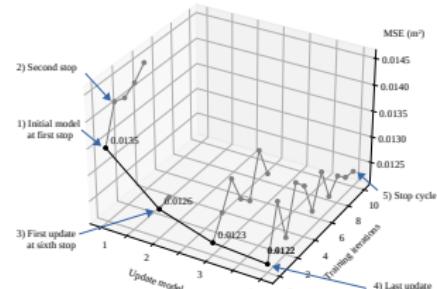
```

Input: MODEL as the input model
Input:  $D_{train}$  as the training data set
Input:  $D_{val}$  as the validation data set
Input:  $N_I$  as the stop patience for iterative training cycle
Input:  $N_E$  as the early stop patience (epochs) for training
Input:  $B_{size}$  as the mini-batch size
Output: MODEL as the full-precision output model

// Initial training and evaluation
1 Train(MODEL,  $D_{train}, D_{val}, N_E, B_{size}$ )
2  $mse_i \leftarrow Evaluate(MODEL, D_{val})$ 
3  $n_I \leftarrow 0$ 
4 while  $n_I < N_I$  do
    // Iterative early stop cycle
    5 Train(MODEL,  $D_{train}, D_{val}, N_E, B_{size}$ )
    6  $mse_v \leftarrow Evaluate(MODEL, D_{val})$ 
    7 if  $mse_v < mse_i$  then
        8     | Update(MODEL)
        9     |  $mse_i \leftarrow mse_v$ 
    10    end
    11    else
    12        | MODEL  $\leftarrow LoadPreviousWeights()$ 
    13        |  $n_I \leftarrow n_I + 1$ 
    14    end
15 end

```

Training with iterative early stop



Training

Algorithm 2: Training with iterative early stop cycle.

```

Input: MODEL as the input model
Input:  $D_{train}$  as the training data set
Input:  $D_{val}$  as the validation data set
Input:  $N_I$  as the stop patience for iterative training cycle
Input:  $N_E$  as the early stop patience (epochs) for training
Input:  $B_{size}$  as the mini-batch size
Output: MODEL as the full-precision output model

// Initial training and evaluation
1 Train(MODEL,  $D_{train}, D_{val}, N_E, B_{size}$ )
2  $mse_i \leftarrow Evaluate(MODEL, D_{val})$ 
3  $n_I \leftarrow 0$ 
4 while  $n_I < N_I$  do
    // Iterative early stop cycle
    5 Train(MODEL,  $D_{train}, D_{val}, N_E, B_{size}$ )
    6  $mse_v \leftarrow Evaluate(MODEL, D_{val})$ 
    7 if  $mse_v < mse_i$  then
        8     | Update(MODEL)
        9     |  $mse_i \leftarrow mse_v$ 
    end
    else
        12     | MODEL  $\leftarrow LoadPreviousWeights()$ 
        13     |  $n_I \leftarrow n_I + 1$ 
    end
15 end

```

Algorithm 3: OnMiniBatchUpdate_Callback.

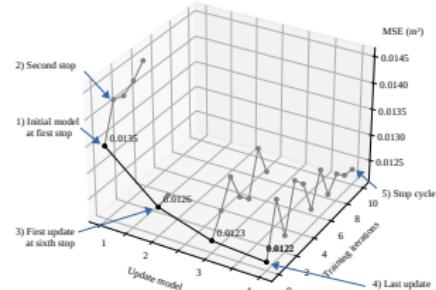
```

Input: MODEL as the full-precision input model
Input:  $E_{size}$  as the target exponent bits size
Input:  $M_{size}$  as the target mantissa bits size
Input:  $D_{train}$  as the training data set
Input:  $D_{val}$  as the validation data set
Input:  $N_{ep}$  as the number of epochs
Input:  $B_{size}$  as the mini-batch size
Output: MODEL as the quantized output model

// Quantize
1 MODEL  $\leftarrow QuantizeTraining(MODEL, E_{size}, M_{size})$ 
2 if 1 < epoch then
    // Update model after first epoch
    3     |  $mse_v \leftarrow Evaluate(MODEL, D_{val})$ 
    4     if  $mse_v < mse_i$  then
        5         | Update(MODEL)
        6         |  $mse_i \leftarrow mse_v$ 
    end
8 end

```

Training with iterative early stop



Training

Algorithm 2: Training with iterative early stop cycle.

```

Input: MODEL as the input model
Input:  $D_{train}$  as the training data set
Input:  $D_{val}$  as the validation data set
Input:  $N_f$  as the stop patience for iterative training cycle
Input:  $N_E$  as the early stop patience (epochs) for training
Input:  $B_{size}$  as the mini-batch size
Output: MODEL as the full-precision output model

// Initial training and evaluation
1 Train(MODEL,  $D_{train}, D_{val}, N_E, B_{size}$ )
2  $mse_i \leftarrow Evaluate(MODEL, D_{val})$ 
3  $n_f \leftarrow 0$ 
4 while  $n_f < N_f$  do
    // Iterative early stop cycle
    5 Train(MODEL,  $D_{train}, D_{val}, N_E, B_{size}$ )
    6  $mse_v \leftarrow Evaluate(MODEL, D_{val})$ 
    7 if  $mse_v < mse_i$  then
        8     | Update(MODEL)
        9     |  $mse_i \leftarrow mse_v$ 
    end
    else
        12     | MODEL  $\leftarrow LoadPreviousWeights()$ 
        13     |  $n_f \leftarrow n_f + 1$ 
    end
15 end

```

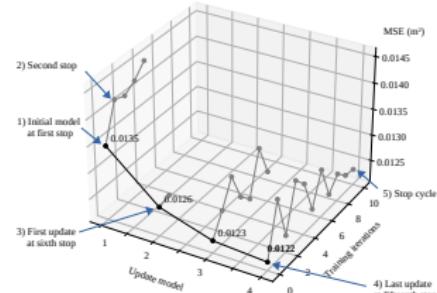
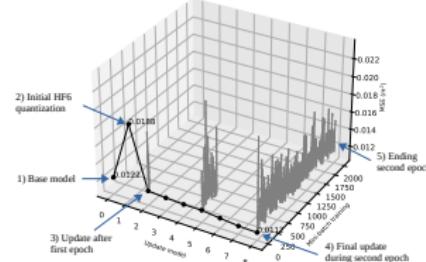
Algorithm 3: OnMiniBatchUpdate_Callback.

```

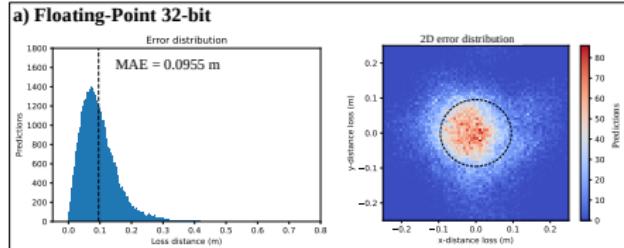
Input: MODEL as the full-precision input model
Input:  $E_{size}$  as the target exponent bits size
Input:  $M_{size}$  as the target mantissa bits size
Input:  $D_{train}$  as the training data set
Input:  $D_{val}$  as the validation data set
Input:  $N_{ep}$  as the number of epochs
Input:  $B_{size}$  as the mini-batch size
Output: MODEL as the quantized output model

// Quantize
1 MODEL  $\leftarrow QuantizeTraining(MODEL, E_{size}, M_{size})$ 
2 if  $1 < epoch$  then
    // Update model after first epoch
    3     |  $mse_v \leftarrow Evaluate(MODEL, D_{val})$ 
    4     if  $mse_v < mse_i$  then
        5         | Update(MODEL)
        6         |  $mse_i \leftarrow mse_v$ 
    end
8 end

```

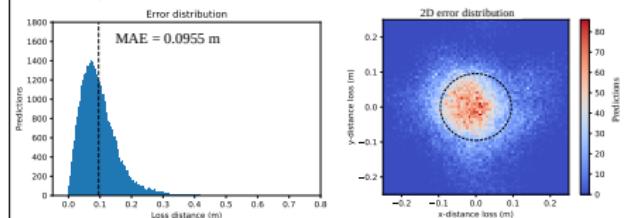
Training with iterative early stop**Quantization-aware training**

Inference

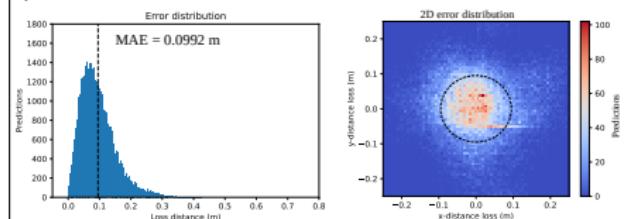


Inference

a) Floating-Point 32-bit

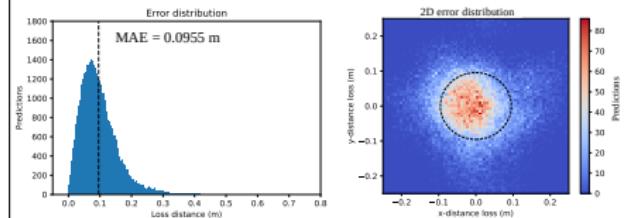


b) Fixed-Point 8-bit

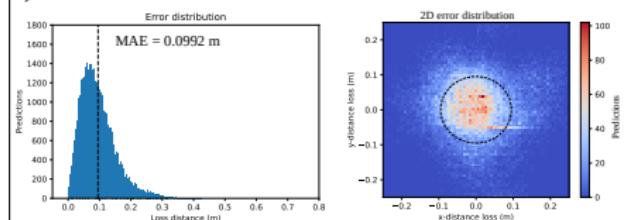


Inference

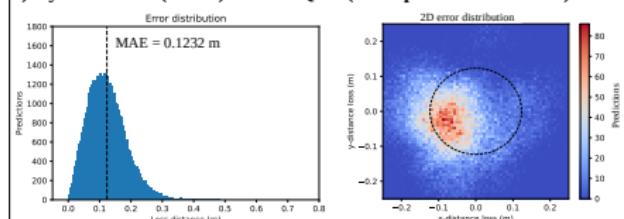
a) Floating-Point 32-bit



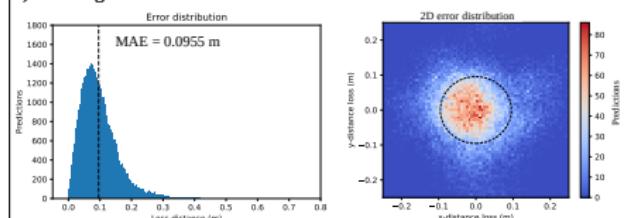
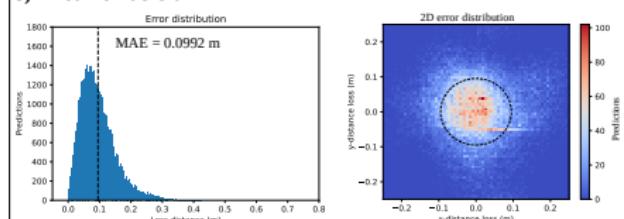
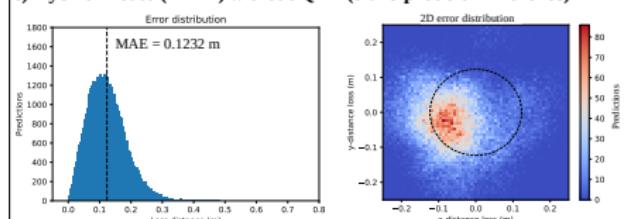
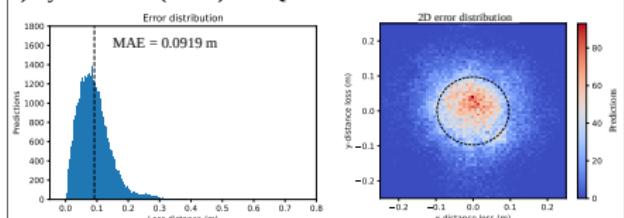
b) Fixed-Point 8-bit



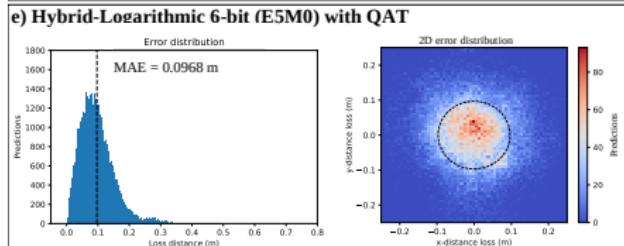
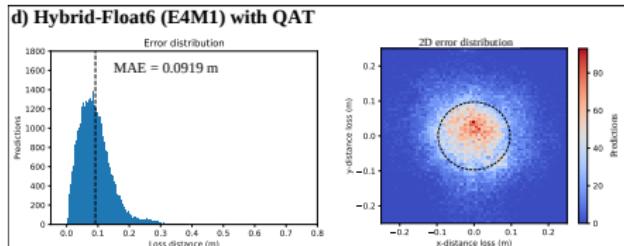
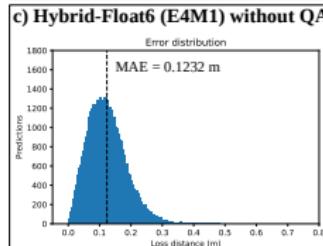
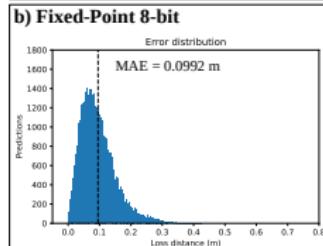
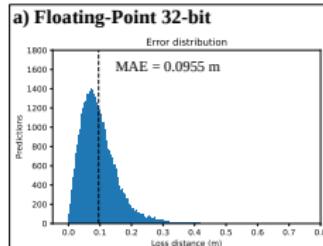
c) Hybrid-Float6 (E4M1) without QAT (trans-precision inference)



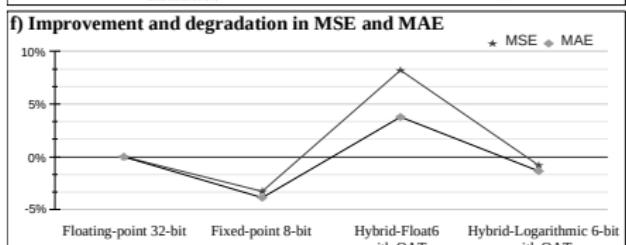
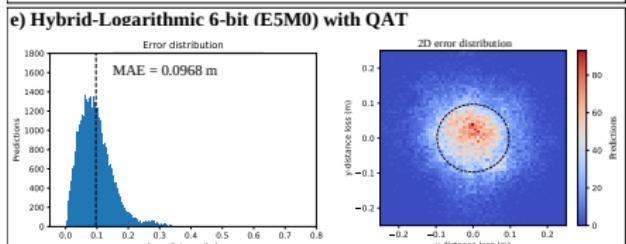
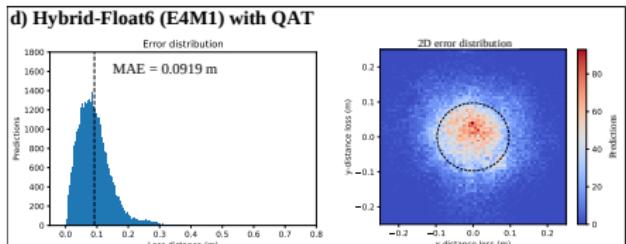
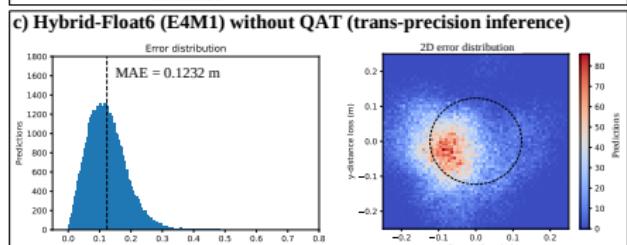
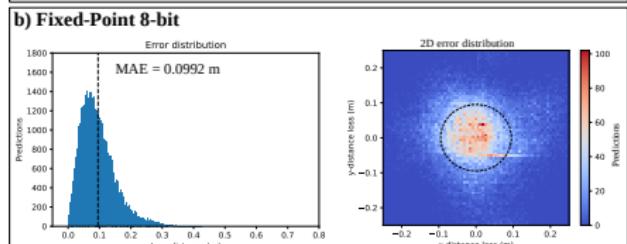
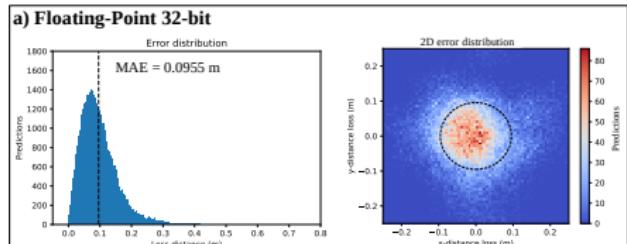
Inference

a) Floating-Point 32-bit**b) Fixed-Point 8-bit****c) Hybrid-Float6 (E4M1) without QAT (trans-precision inference)****d) Hybrid-Float6 (E4M1) with QAT**

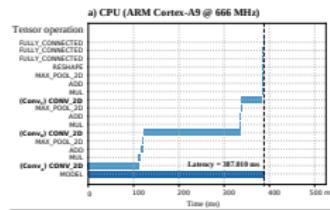
Inference



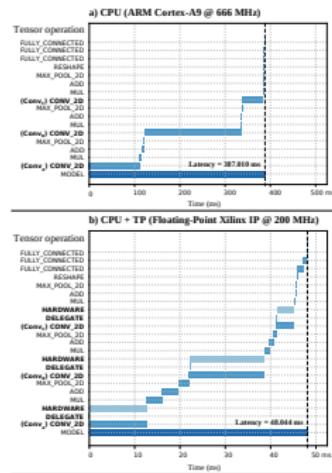
Inference



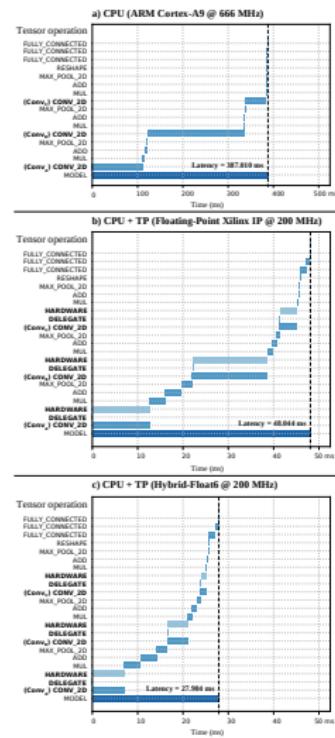
Acceleration, Power Reduction, and Hardware Resources



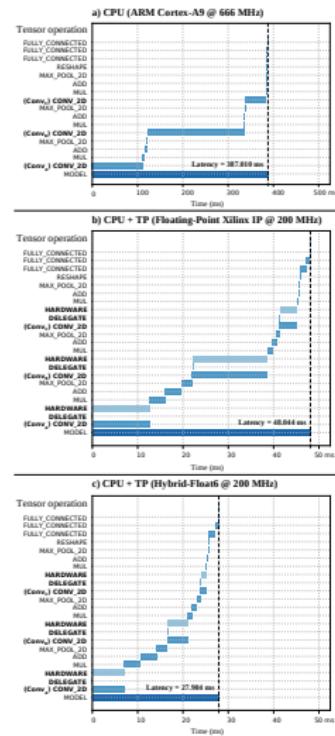
Acceleration, Power Reduction, and Hardware Resources



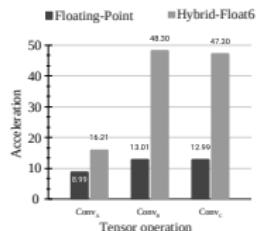
Acceleration, Power Reduction, and Hardware Resources



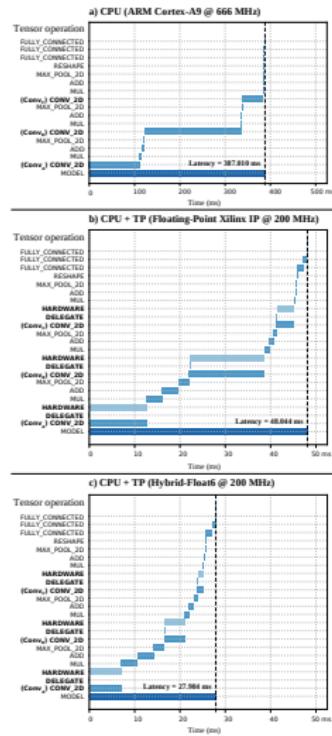
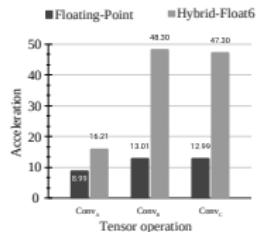
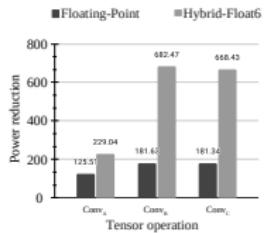
Acceleration, Power Reduction, and Hardware Resources



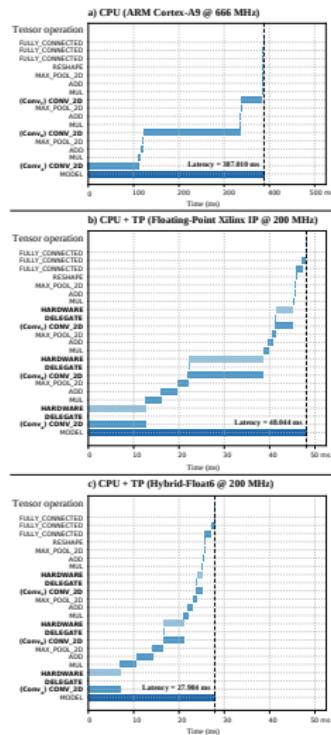
Acceleration vs. CPU



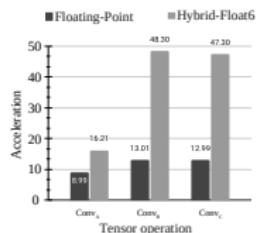
Acceleration, Power Reduction, and Hardware Resources

**Acceleration vs. CPU****Power reduction vs. CPU**

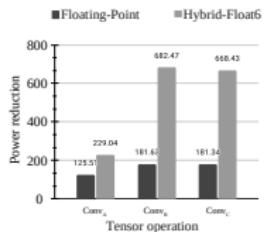
Acceleration, Power Reduction, and Hardware Resources



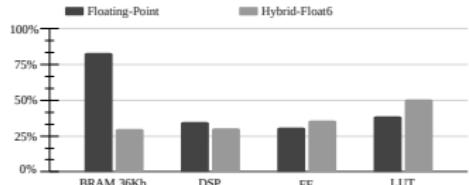
Acceleration vs. CPU



Power reduction vs. CPU



Hardware resource utilization.



Comparison with Related Work

Comparison with related work with floating-point and fixed-point.

Platform	Chunsheng <i>et al.</i> [1]	Chen <i>et al.</i> [2]	BFP [3]	Paolo <i>et al.</i> [4]	This work
Device	XC7VX690T	XC7K325T	XC7VX690T	XC7Z007S	XC7Z007S
Year	2017	2019	2019	2019	2023
Dev. kit cost	\$7,494	\$1,299	\$7,494	\$89	\$89
Format (activation/weight)	FP 16-bit	FP 8-bit / 8-bit	FP 16-bit / 8-bit	INT 16-bit	FP 32-bit / 6-bit
Frequency (MHz)	200	200	200	80	200
Peak power efficiency (GFLOP/s/W)	18.72	115.40	82.88	2.98	5.74
Peak throughput (GFLOP/s)	202.42	1086.8	760.83	10.62	0.482
Wall plug power (W)	10.81	9.42	9.18	2.5	2.3
BRAM 36Kb utilization	196.5	234.5	913	44	15
DSP utilization	1728	768	1027	54	20

- [1] Chunsheng Mei, Zhenyu Liu, Yue Niu, Xiangyang Ji, Wei Zhou, and Dongsheng Wang. A 200mhz 202.4 gflops@ 10.8 w vgg16 accelerator in xilinx vx690t. In *2017 IEEE Global Conference on Signal and Information Processing (GlobalSIP)*, pages 784–788. IEEE, 2017.
- [2] Chen Wu, Mingyu Wang, Xinyuan Chu, Kun Wang, and Lei He. Low-precision floating-point arithmetic for high-performance fpga-based cnn acceleration. *ACM Transactions on Reconfigurable Technology and Systems (TRETS)*, 15(1):1–21, 2021.
- [3] Xiaocong Lian, Zhenyu Liu, Zhourui Song, Jiwu Dai, Wei Zhou, and Xiangyang Ji. High-performance fpga-based cnn accelerator with block-floating-point arithmetic. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 27(8):1874–1885, 2019.
- [4] Paolo Meloni, Antonio Garufi, Gianfranco Deriu, Marco Carreras, and Daniela Loi. Cnn hardware acceleration on a low-power and low-cost apsoc. In *2019 Conference on Design and Architectures for Signal and Image Processing (DASIP)*, pages 7–12. IEEE, 2019.

1 State-of-the-Art

2 Hybrid 8-bit Floating-Point and 4-bit Logarithmic Computation

3 Hybrid 6-bit Floating-Point Computation

4 Conclusions and Future Research

Conclusions and Future Research

Conclusions

- Developed low-power neural network accelerators optimizing energy and computational efficiency.

Conclusions and Future Research

Conclusions

- Developed low-power neural network accelerators optimizing energy and computational efficiency.
- Introduced hybrid computation models (8-bit FP and 4-bit Logarithmic), enhancing performance with minimal accuracy loss.

Conclusions and Future Research

Conclusions

- Developed low-power neural network accelerators optimizing energy and computational efficiency.
- Introduced hybrid computation models (8-bit FP and 4-bit Logarithmic), enhancing performance with minimal accuracy loss.
- Demonstrated significant improvements in hardware efficiency and application through advanced quantization strategies.

Conclusions and Future Research

Conclusions

- Developed low-power neural network accelerators optimizing energy and computational efficiency.
- Introduced hybrid computation models (8-bit FP and 4-bit Logarithmic), enhancing performance with minimal accuracy loss.
- Demonstrated significant improvements in hardware efficiency and application through advanced quantization strategies.

Future Research

- Explore further optimization of gradient formats to reduce memory usage and energy consumption.

Conclusions and Future Research

Conclusions

- Developed low-power neural network accelerators optimizing energy and computational efficiency.
- Introduced hybrid computation models (8-bit FP and 4-bit Logarithmic), enhancing performance with minimal accuracy loss.
- Demonstrated significant improvements in hardware efficiency and application through advanced quantization strategies.

Future Research

- Explore further optimization of gradient formats to reduce memory usage and energy consumption.
- Extend methodologies to broader AI/ML applications, testing different computational structures and memory strategies.

Conclusions and Future Research

Conclusions

- Developed low-power neural network accelerators optimizing energy and computational efficiency.
- Introduced hybrid computation models (8-bit FP and 4-bit Logarithmic), enhancing performance with minimal accuracy loss.
- Demonstrated significant improvements in hardware efficiency and application through advanced quantization strategies.

Future Research

- Explore further optimization of gradient formats to reduce memory usage and energy consumption.
- Extend methodologies to broader AI/ML applications, testing different computational structures and memory strategies.
- Investigate the implementation of reduced precision formats in high-demand data center environments.

Publications

Journal Articles

- **Yarib Nevarez**, David Rotermund, Klaus R Pawelzik, and Alberto Garcia-Ortiz, "Accelerating Spike-by-Spike Neural Networks on FPGA With Hybrid Custom Floating-Point and Logarithmic Dot-Product Approximation," *IEEE Access*, vol. 9, pp. 80603–80620, May 2021, doi: 10.1109/ACCESS.2021.3085216.
- **Yarib Nevarez**, Andreas Beering, Amir Najafi, Ardalan Najafi, Wanli Yu, Yizhi Chen, Karl-Ludwig Krieger, and Alberto Garcia-Ortiz, "CNN Sensor Analytics With Hybrid-Float6 Quantization on Low-Power Embedded FPGAs," *IEEE Access*, vol. 11, pp. 4852–4868, January 2023, doi: 10.1109/ACCESS.2023.3235866.

Conference Proceedings

- **Yarib Nevarez**, Alberto Garcia-Ortiz, David Rotermund, and Klaus R Pawelzik, "Accelerator framework of spike-by-spike neural networks for inference and incremental learning in embedded systems," 2020 9th International Conference on Modern Circuits and Systems Technologies (MOCAST), Bremen, 2020, pp. 1–5, doi: 10.1109/MOCAST49295.2020.9200288.
- Wanli Yu, Ardalan Najafi, **Yarib Nevarez**, Yanqiu Huang and Alberto Garcia-Ortiz, "TAAC: Task Allocation Meets Approximate Computing for Internet of Things," 2020 IEEE International Symposium on Circuits and Systems (ISCAS), Sevilla, 2020, pp. 1-5, doi: 10.1109/ISCAS45731.2020.9180895.

Publications

- Amir Najafi, Ardalan Najafi, **Yarib Nevarez** and Alberto Garcia-Ortiz, "Learning-Based On-Chip Parallel Interconnect Delay Estimation," 2022 11th International Conference on Modern Circuits and Systems Technologies (MOCAST), Bremen, 2022, pp. 1–5, doi: 10.1109/MOCAST49295.2020.9200288.
- Yizhi Chen, **Yarib Nevarez**, Zhonghai Lu, and Alberto Garcia-Ortiz, "Accelerating Non-Negative Matrix Factorization on Embedded FPGA with Hybrid Logarithmic Dot-Product Approximation," 2022 IEEE 15th International Symposium on Embedded Multicore/Many-core Systems-on-Chip (MCSoC), Malaysia, 2022, pp. 239–246, doi: 10.1109/MCSoC57363.2022.00070.
- Ardalan Najafi, Wanli Yu, **Yarib Nevarez**, Amir Najafi, Andreas Beering, Karl-Ludwig Krieger, and Alberto Garcia-Ortiz, "Acoustic Emission Source Localization using Approximate Discrete Wavelet Transform," 2023 12th International Conference on Modern Circuits and Systems Technologies (MOCAST), Bremen, 2023, pp. 1–5, doi: 10.1109/MOCAST57943.2023.10176952.