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

Limitations of State-of-the-Art

- **Spike-by-Spike Neural Networks:**

Limitations of State-of-the-Art

- **Spike-by-Spike Neural Networks:**
  - Quantization Challenges: Struggles with reduced number representations (8-bit and 4-bit)

Limitations of State-of-the-Art

- **Spike-by-Spike Neural Networks:**
  - Quantization Challenges: Struggles with reduced number representations (8-bit and 4-bit)
  - Deployment Frameworks: Lack of deployment strategies for IoT and TinyML applications

Limitations of State-of-the-Art

- **Spike-by-Spike Neural Networks:**
  - Quantization Challenges: Struggles with reduced number representations (8-bit and 4-bit)
  - Deployment Frameworks: Lack of deployment strategies for IoT and TinyML applications
  - Design Methodologies: Lack of effective low-power accelerator design methodologies

Limitations of State-of-the-Art

- **Spike-by-Spike Neural Networks:**
  - Quantization Challenges: Struggles with reduced number representations (8-bit and 4-bit)
  - Deployment Frameworks: Lack of deployment strategies for IoT and TinyML applications
  - Design Methodologies: Lack of effective low-power accelerator design methodologies
- **Low-Power CNN Accelerators:**

Limitations of State-of-the-Art

- **Spike-by-Spike Neural Networks:**
  - Quantization Challenges: Struggles with reduced number representations (8-bit and 4-bit)
  - Deployment Frameworks: Lack of deployment strategies for IoT and TinyML applications
  - Design Methodologies: Lack of effective low-power accelerator design methodologies
- **Low-Power CNN Accelerators:**
  - Floating-Point Design: Literature gap in floating-point methodologies for IoT and TinyML applications

Limitations of State-of-the-Art

- **Spike-by-Spike Neural Networks:**
  - Quantization Challenges: Struggles with reduced number representations (8-bit and 4-bit)
  - Deployment Frameworks: Lack of deployment strategies for IoT and TinyML applications
  - Design Methodologies: Lack of effective low-power accelerator design methodologies
- **Low-Power CNN Accelerators:**
  - Floating-Point Design: Literature gap in floating-point methodologies for IoT and TinyML applications
  - Reproducibility: Deficiency in reproducible research, limiting validation and adoption

Limitations of State-of-the-Art

- **Spike-by-Spike Neural Networks:**
  - Quantization Challenges: Struggles with reduced number representations (8-bit and 4-bit)
  - Deployment Frameworks: Lack of deployment strategies for IoT and TinyML applications
  - Design Methodologies: Lack of effective low-power accelerator design methodologies
- **Low-Power CNN Accelerators:**
  - Floating-Point Design: Literature gap in floating-point methodologies for IoT and TinyML applications
  - Reproducibility: Deficiency in reproducible research, limiting validation and adoption
- **Low-Power Accelerators with Aggressive Quantization:**

Limitations of State-of-the-Art

- **Spike-by-Spike Neural Networks:**
  - Quantization Challenges: Struggles with reduced number representations (8-bit and 4-bit)
  - Deployment Frameworks: Lack of deployment strategies for IoT and TinyML applications
  - Design Methodologies: Lack of effective low-power accelerator design methodologies
- **Low-Power CNN Accelerators:**
  - Floating-Point Design: Literature gap in floating-point methodologies for IoT and TinyML applications
  - Reproducibility: Deficiency in reproducible research, limiting validation and adoption
- **Low-Power Accelerators with Aggressive Quantization:**
  - Computational Efficiency vs. Accuracy: Enhances efficiency but often at the expense of significant accuracy degradation in complex tasks

Limitations of State-of-the-Art

- **Spike-by-Spike Neural Networks:**
  - Quantization Challenges: Struggles with reduced number representations (8-bit and 4-bit)
  - Deployment Frameworks: Lack of deployment strategies for IoT and TinyML applications
  - Design Methodologies: Lack of effective low-power accelerator design methodologies
- **Low-Power CNN Accelerators:**
  - Floating-Point Design: Literature gap in floating-point methodologies for IoT and TinyML applications
  - Reproducibility: Deficiency in reproducible research, limiting validation and adoption
- **Low-Power Accelerators with Aggressive Quantization:**
  - Computational Efficiency vs. Accuracy: Enhances efficiency but often at the expense of significant accuracy degradation in complex tasks
  - Mission-Critical Applications: Often unsuitable for applications requiring high reliability, safety, and quality-of-result

Limitations of State-of-the-Art

- **Spike-by-Spike Neural Networks:**
  - Quantization Challenges: Struggles with reduced number representations (8-bit and 4-bit)
  - Deployment Frameworks: Lack of deployment strategies for IoT and TinyML applications
  - Design Methodologies: Lack of effective low-power accelerator design methodologies
- **Low-Power CNN Accelerators:**
  - Floating-Point Design: Literature gap in floating-point methodologies for IoT and TinyML applications
  - Reproducibility: Deficiency in reproducible research, limiting validation and adoption
- **Low-Power Accelerators with Aggressive Quantization:**
  - Computational Efficiency vs. Accuracy: Enhances efficiency but often at the expense of significant accuracy degradation in complex tasks
  - Mission-Critical Applications: Often unsuitable for applications requiring high reliability, safety, and quality-of-result
  - Compatibility and Portability: Faces challenges in compatibility and portability across different computing platforms and ML frameworks

Spike-by-Spike Neural Network

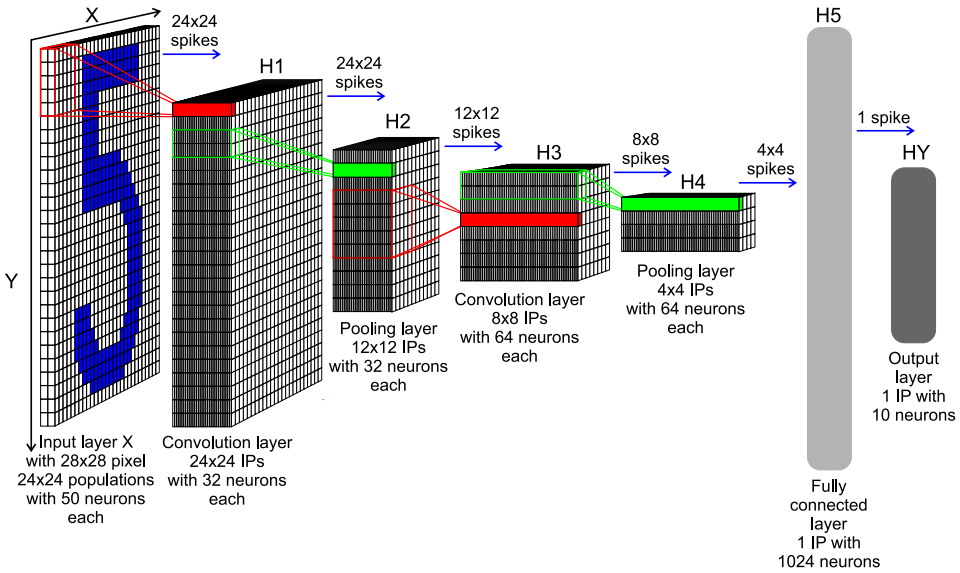


Figure: Spike-by-Spike (SbS) neural network architecture for handwritten digit classification task

Spike-by-Spike Neural Network

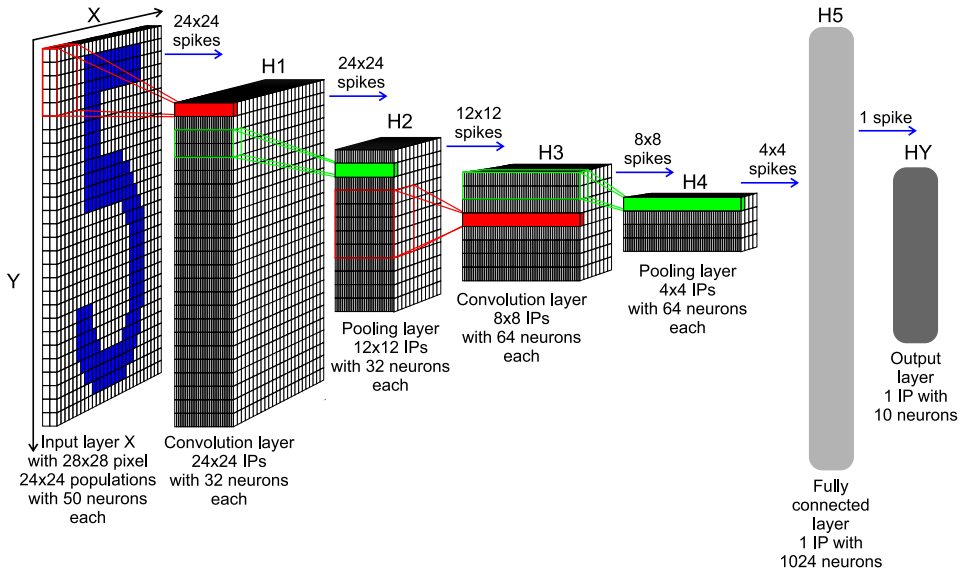


Figure: Spike-by-Spike (SbS) neural network architecture for handwritten digit classification task

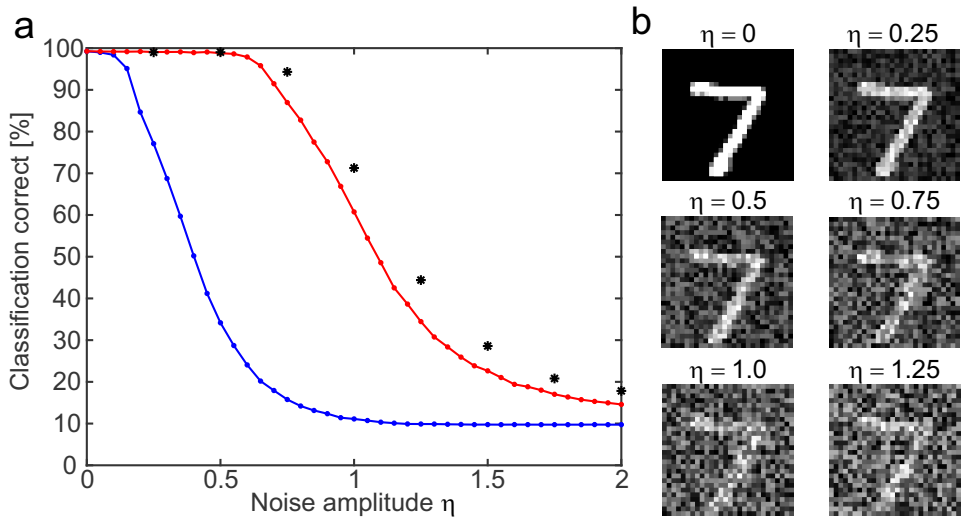


Figure: Performance classification of SbS NN versus equivalent CNN

Spike-by-Spike Inference

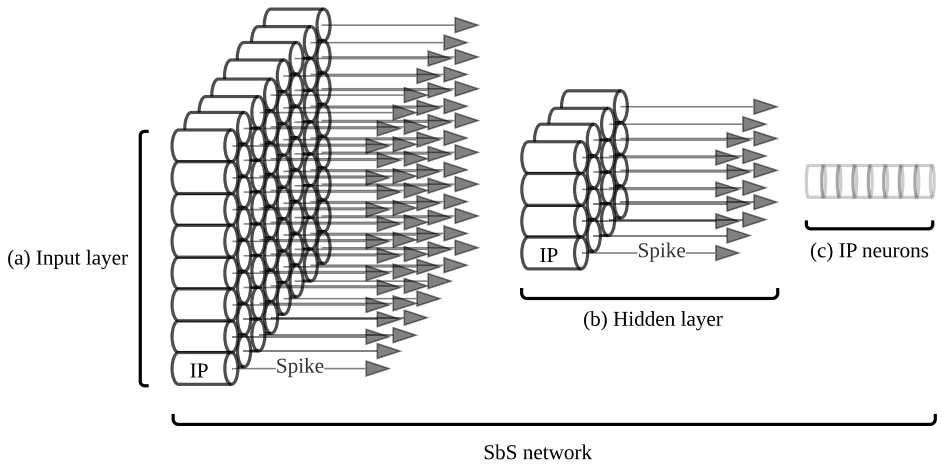


Figure: SbS inference population (IP) as independent computational entities

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

HW/SW Co-Design and Deployment Framework

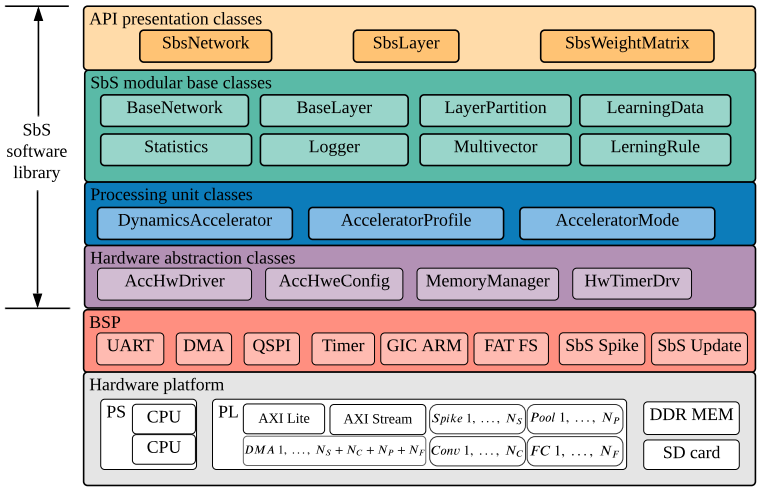


Figure: System-level overview of the embedded software architecture

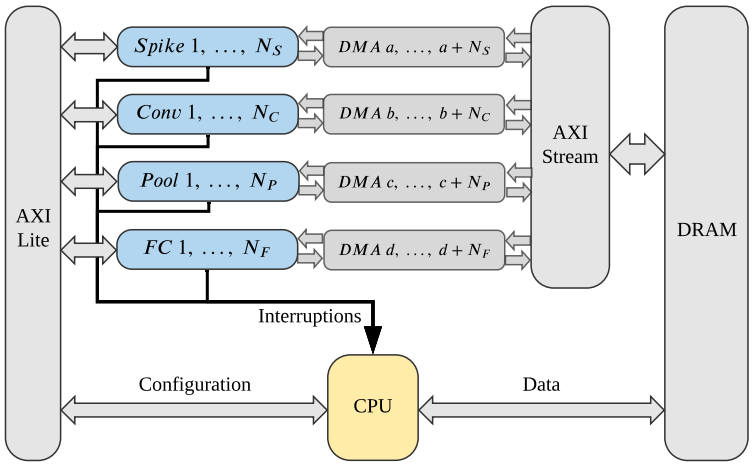


Figure: System-level hardware architecture with scalable number of heterogeneous processing units

Processing Unit

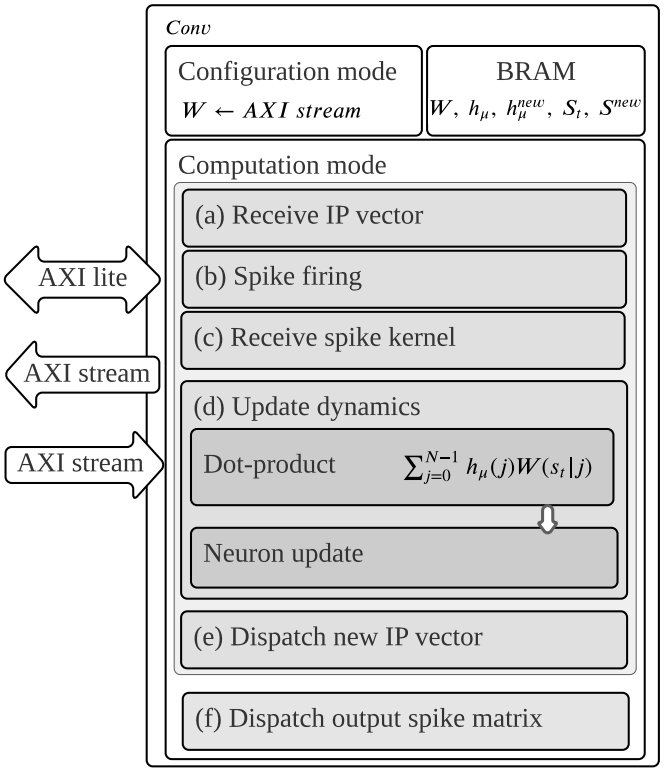


Figure: Convolution processing unit

Processing Unit

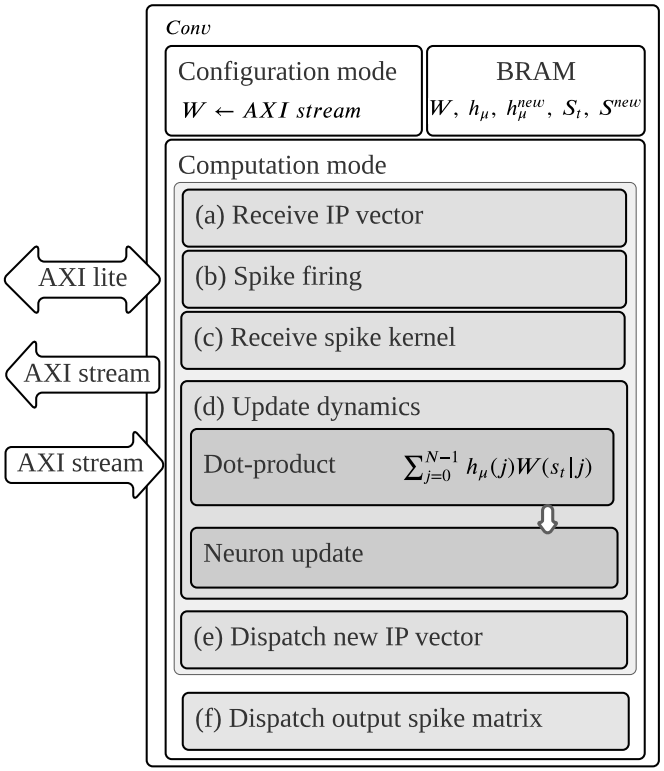


Figure: Convolution processing unit

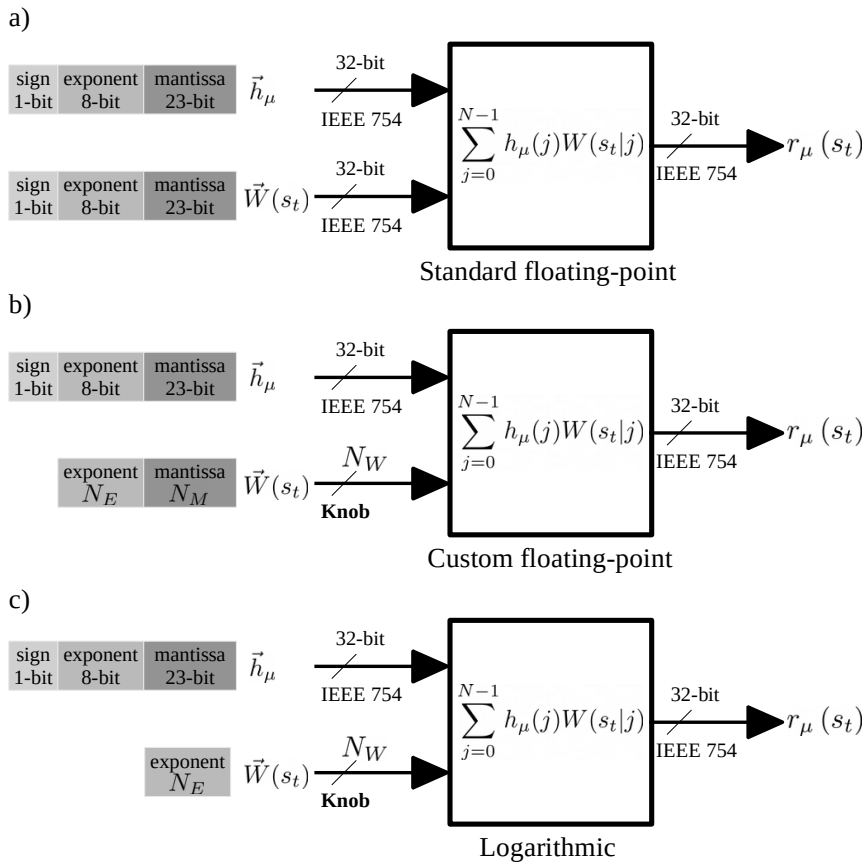


Figure: Dot-product hardware module

Hybrid Dot-Product Approximation

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

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

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

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

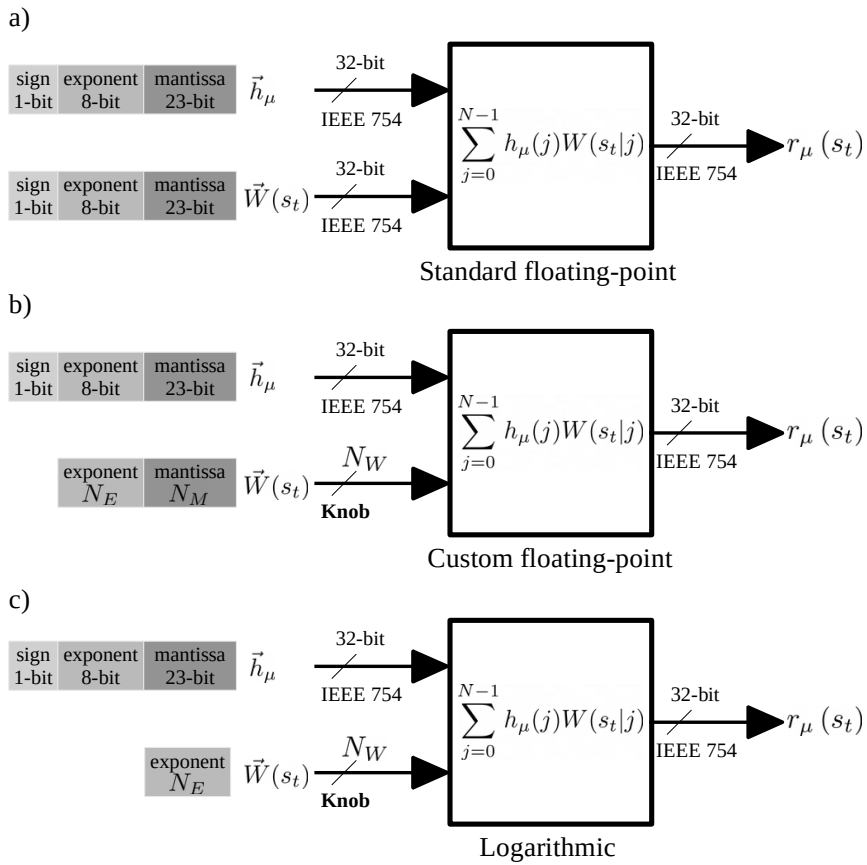


Figure: Dot-product hardware module

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

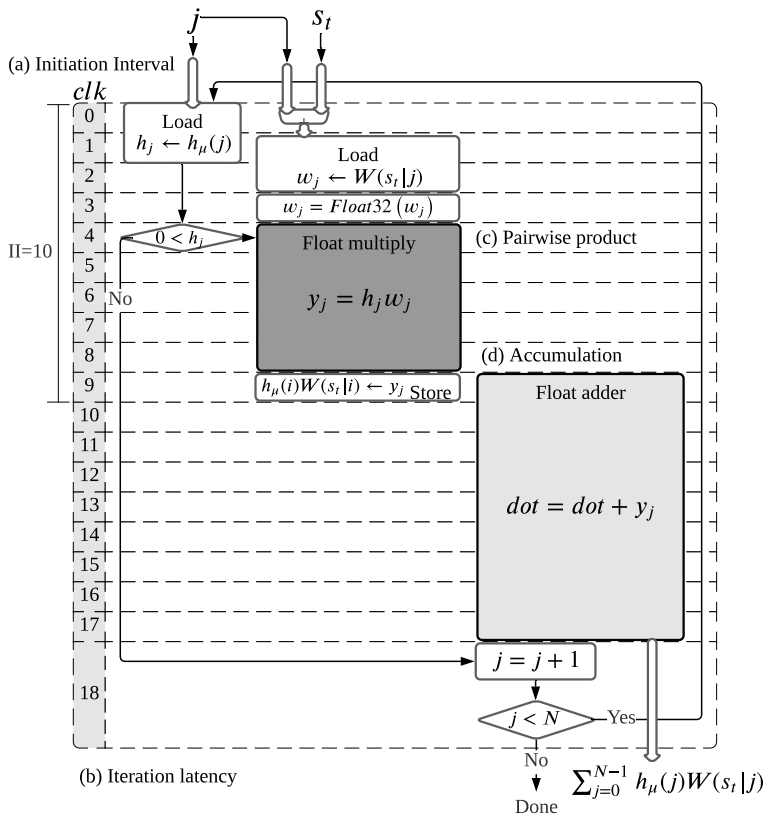


Figure: Dot-product hardware module with standard floating-point computation

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

Dot-Product with Hybrid Custom Floating-Point Approximation

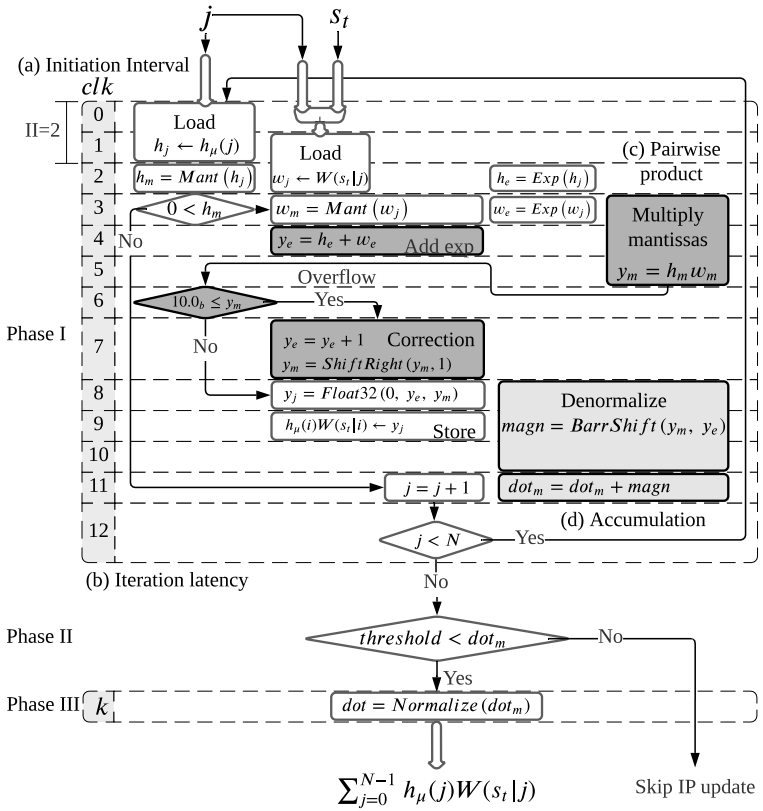


Figure: Dot-product hardware module with hybrid custom floating-point approximation

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

Dot-product with Hybrid Logarithmic Approximation

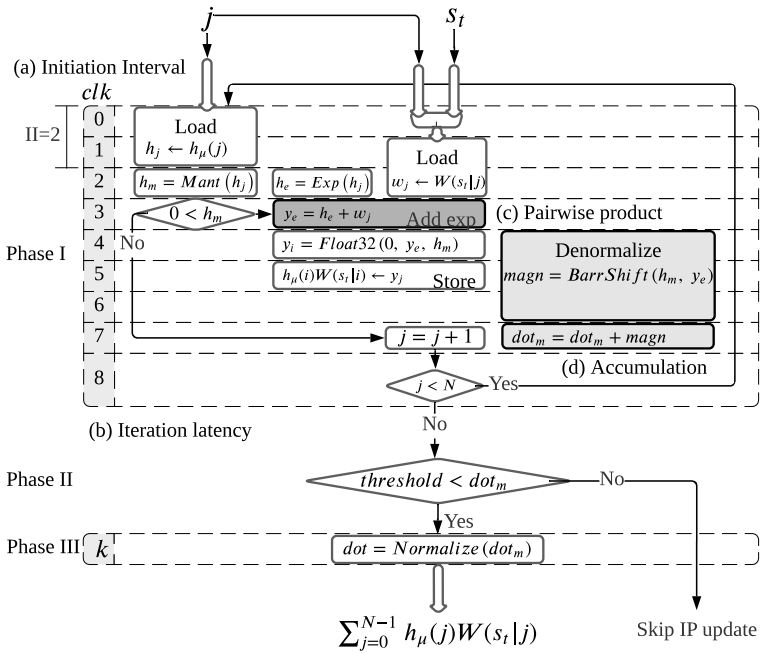


Figure: Dot-product hardware module with hybrid logarithmic approximation

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

Deployment with Standard Floating-Point

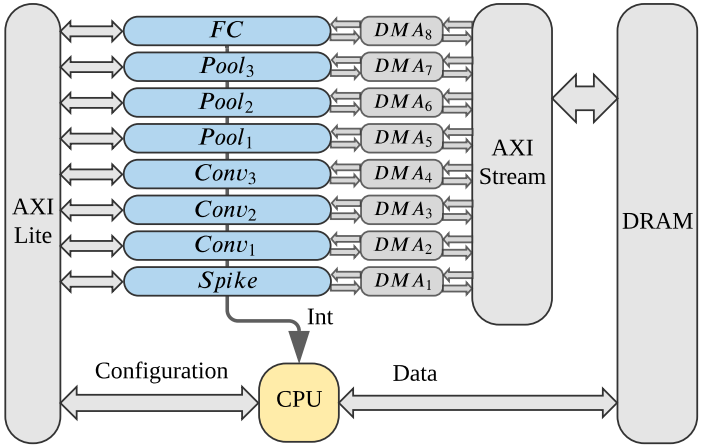


Figure: System overview of the top-level architecture with 8 processing units

Deployment with Standard Floating-Point

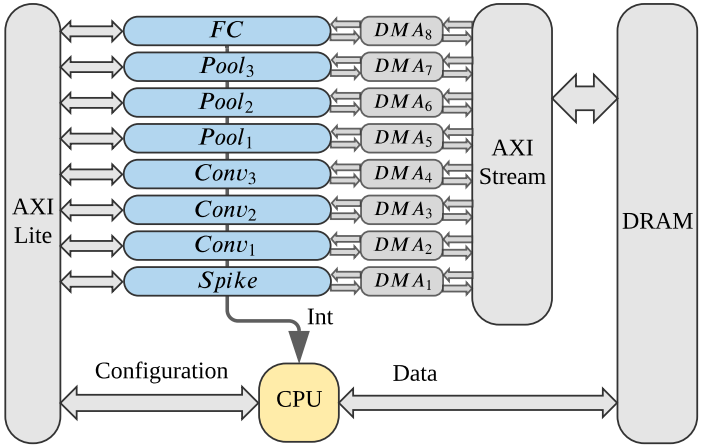


Figure: System overview of the top-level architecture with 8 processing units

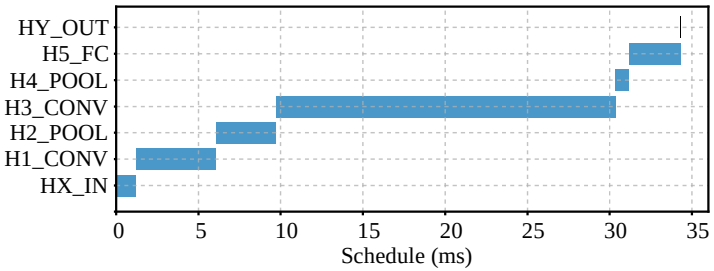


Figure: Computation on embedded CPU

Deployment with Standard Floating-Point

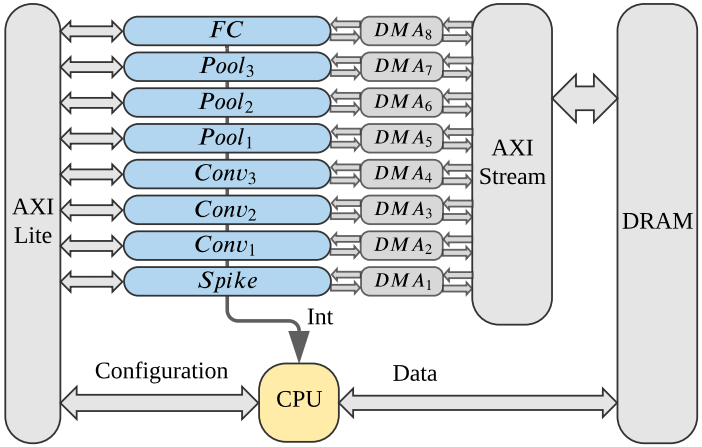


Figure: System overview of the top-level architecture with 8 processing units

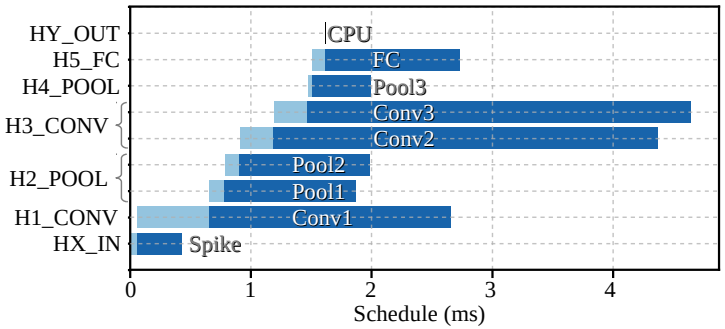


Figure: Performance of processing units with standard floating-point with **acceleration of 10.7X**

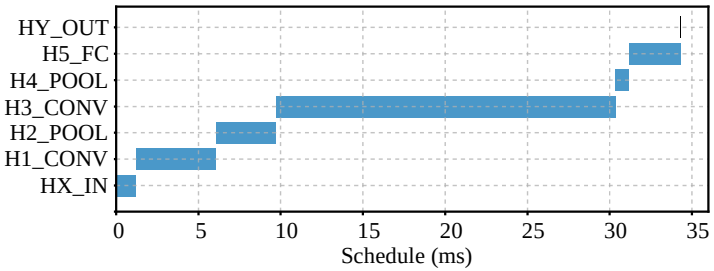


Figure: Computation on embedded CPU

Deployment with Standard Floating-Point

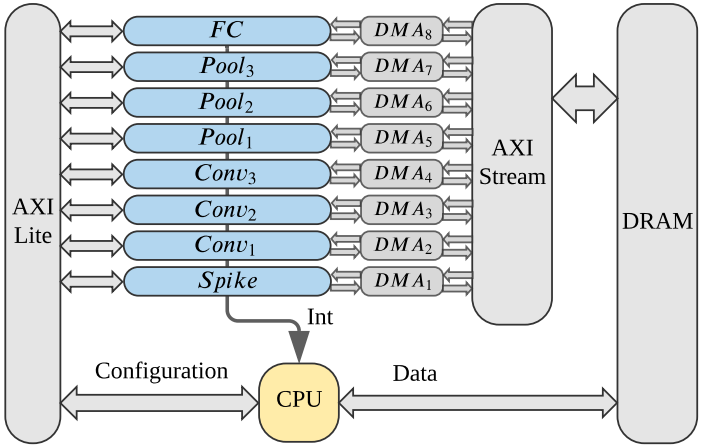


Figure: System overview of the top-level architecture with 8 processing units

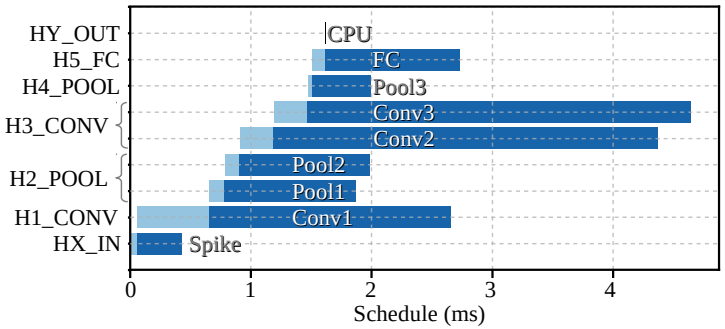


Figure: Performance of processing units with standard floating-point with **acceleration of 10.7X**

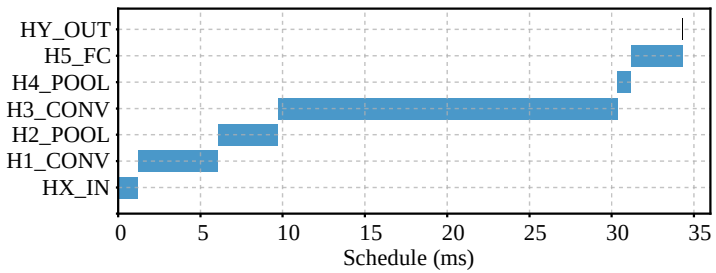


Figure: Computation on embedded CPU

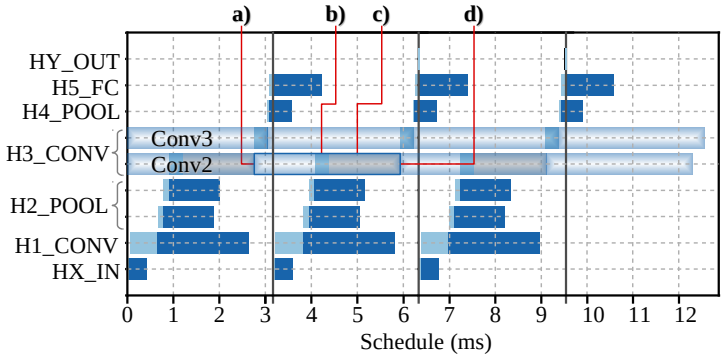


Figure: Performance bottleneck of cyclic computation on processing units with standard floating-point

Deployment with Custom Floating-Point

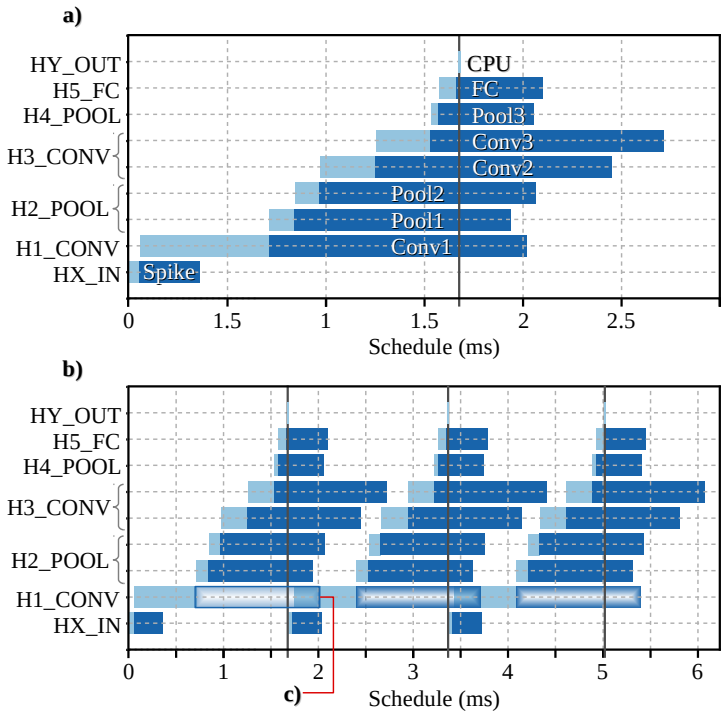


Figure: Performance on processing units with hybrid **8-bit floating-point** with **acceleration of 20.5X**

Deployment with Custom Floating-Point

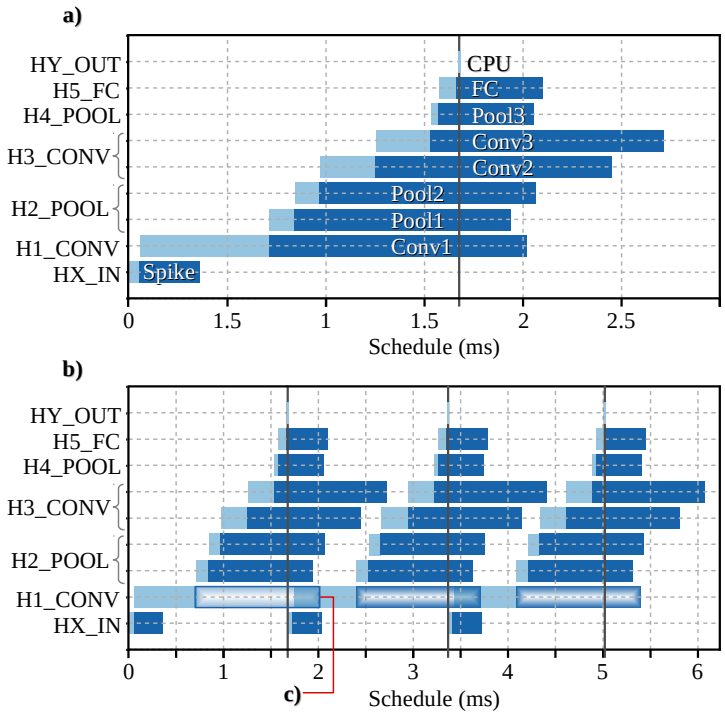


Figure: Performance on processing units with hybrid **8-bit floating-point** with **acceleration of 20.5X**

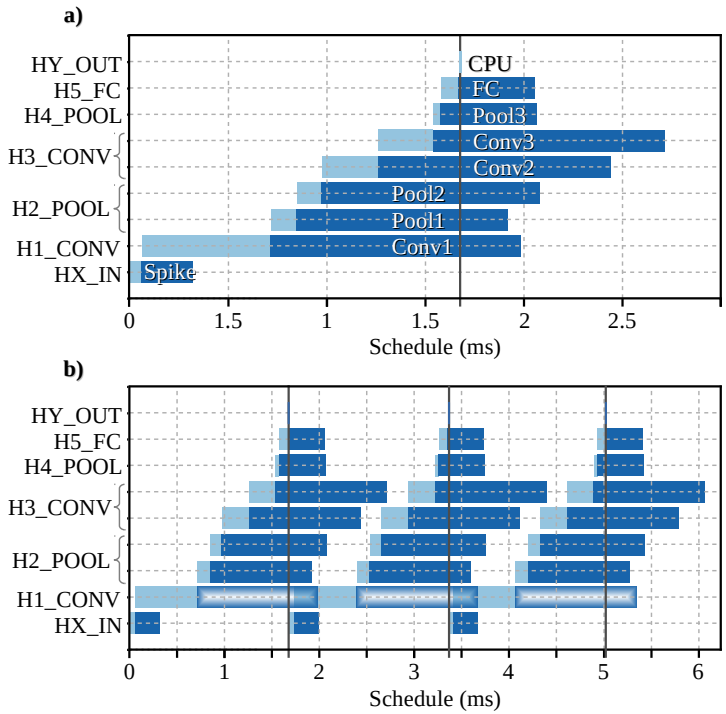


Figure: Performance of processing units with hybrid **4-bit logarithmic** with **acceleration of 20.5X**

Quantization Impact: Noise Tolerance

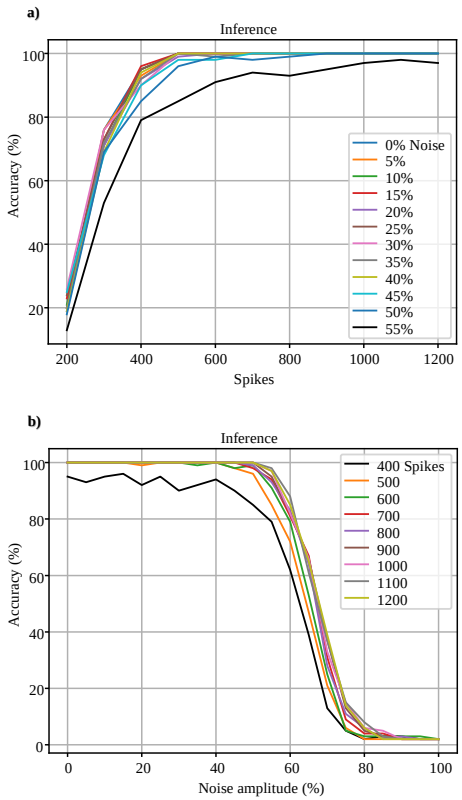


Figure: Noise tolerance with **32-bit floating-point**

Quantization Impact: Noise Tolerance

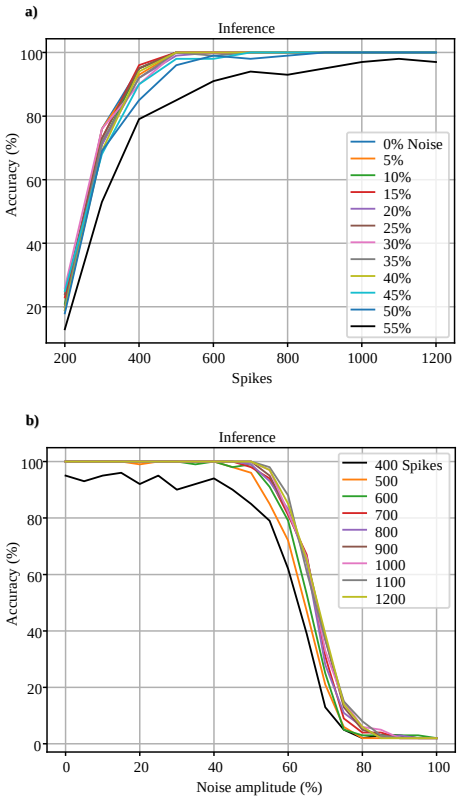


Figure: Noise tolerance with **32-bit floating-point**

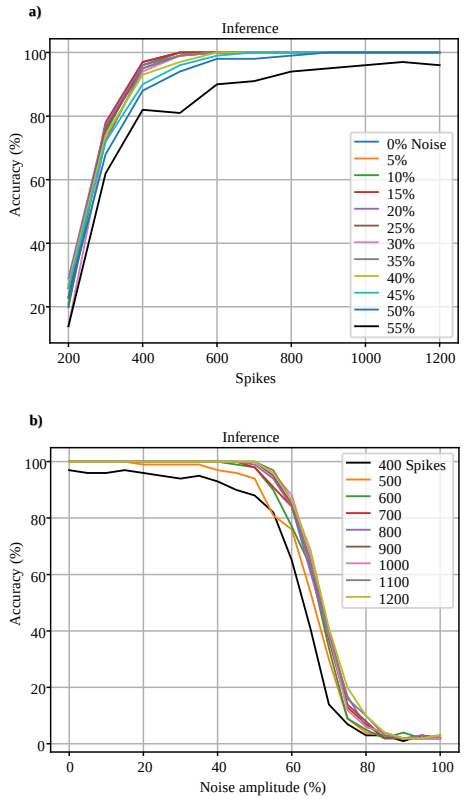


Figure: Noise tolerance with hybrid **8-bit floating-point**

Quantization Impact: Noise Tolerance

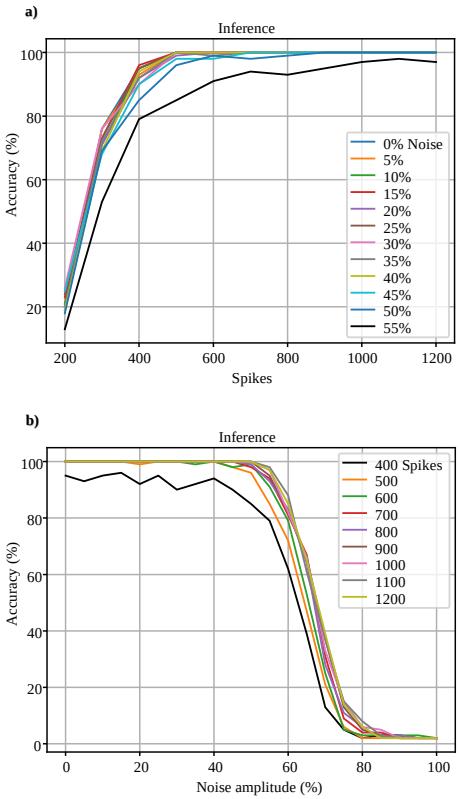


Figure: Noise tolerance with **32-bit floating-point**

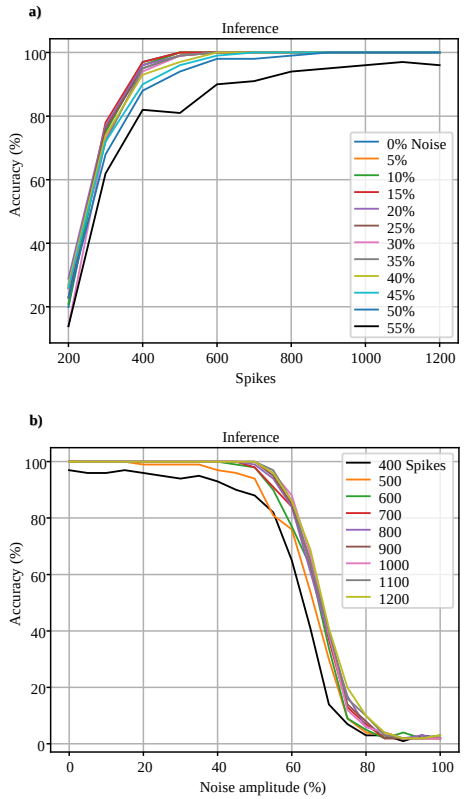


Figure: Noise tolerance with hybrid **8-bit floating-point**

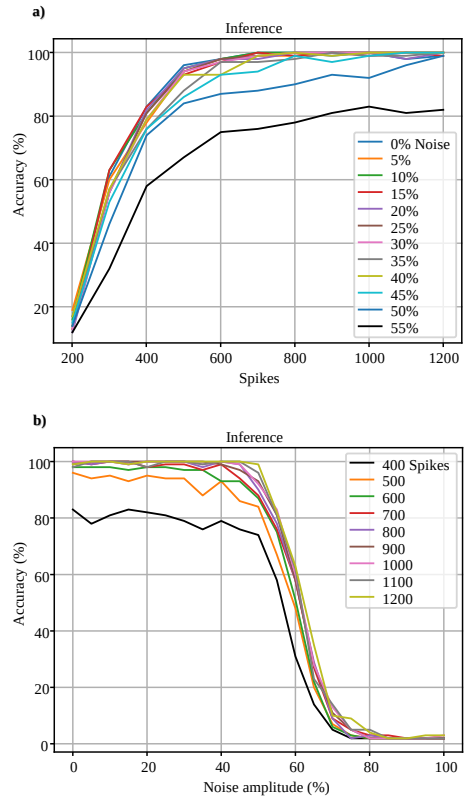


Figure: Noise tolerance with hybrid **4-bit logarithmic**

Convolution Operation

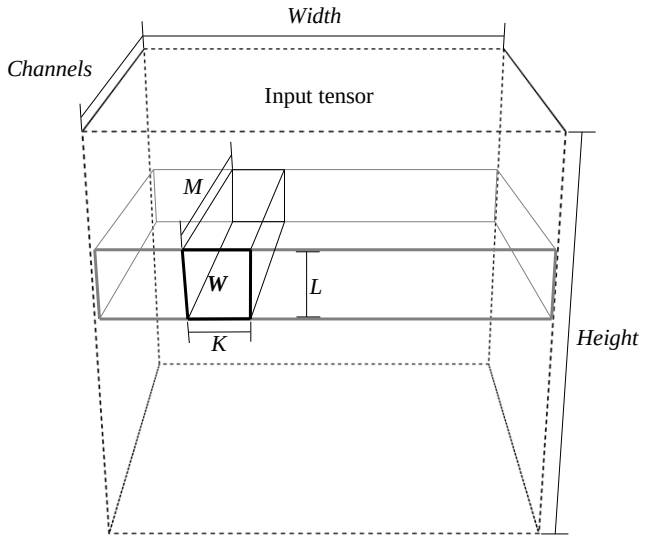


Figure: Two dimensional convolution operation

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

HW/SW Co-Design and Deployment Framework

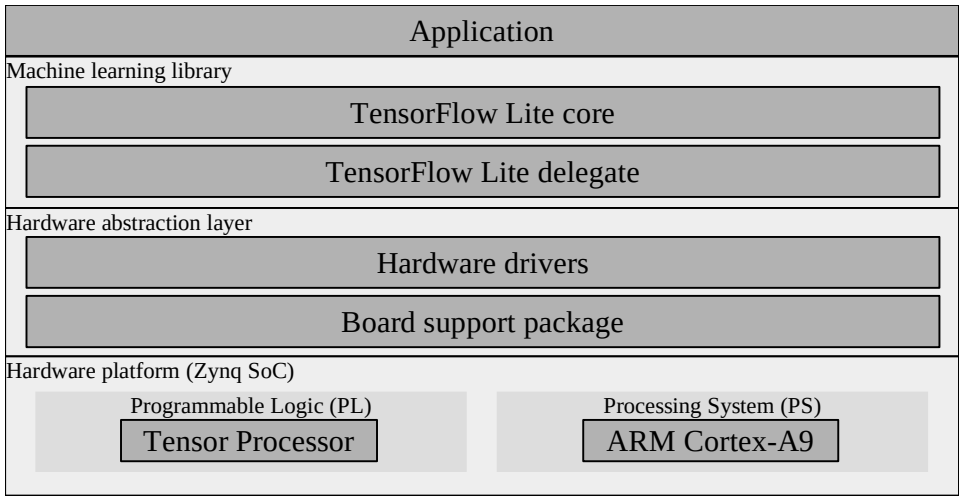


Figure: High level embedded software architecture

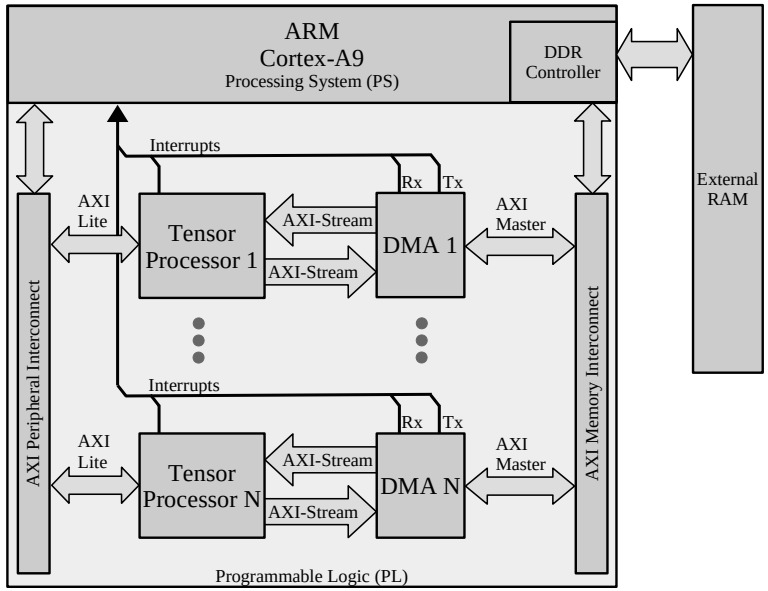


Figure: Base embedded system architecture

Tensor Processor

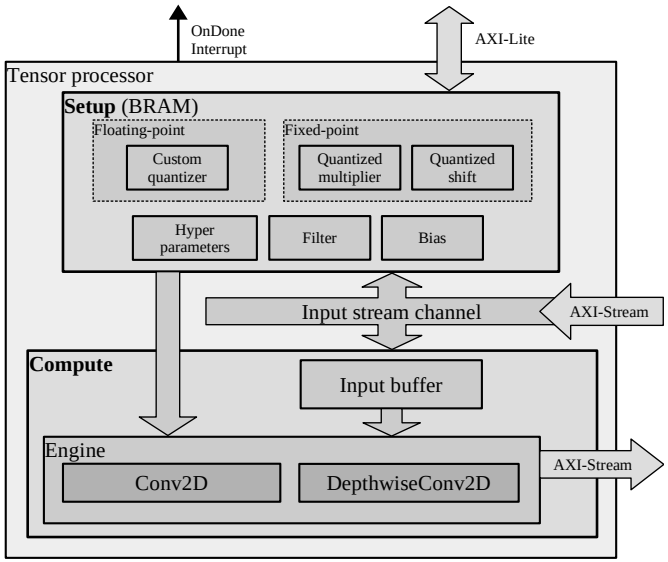


Figure: High level architecture of tensor processor

Tensor Processor

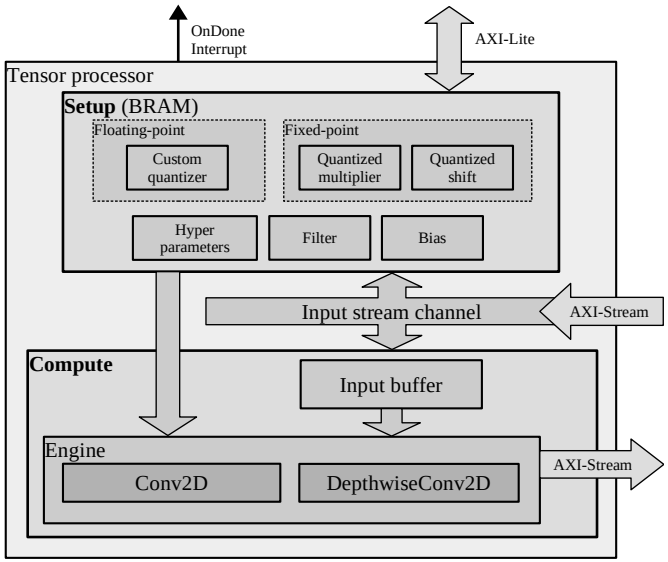


Figure: High level architecture of tensor processor

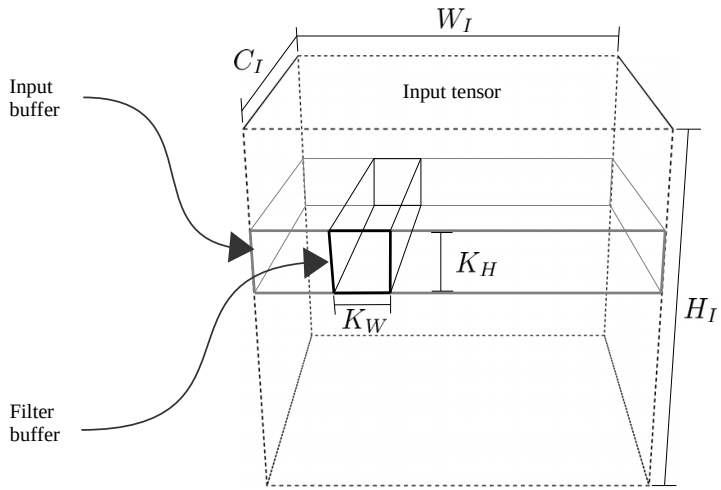


Figure: On-chip memory buffers

On-Chip Memory

$$TP_M = TP_B + V_M \tag{5}$$

$$TP_B = Input_M + Filter_M + Bias_M \tag{6}$$

$$Input_M = K_H W_I C_I BitSize_I \tag{7}$$

$$Filter_M = C_I K_W K_H C_O BitSize_F \tag{8}$$

$$Bias_M = C_O BitSize_B \tag{9}$$

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

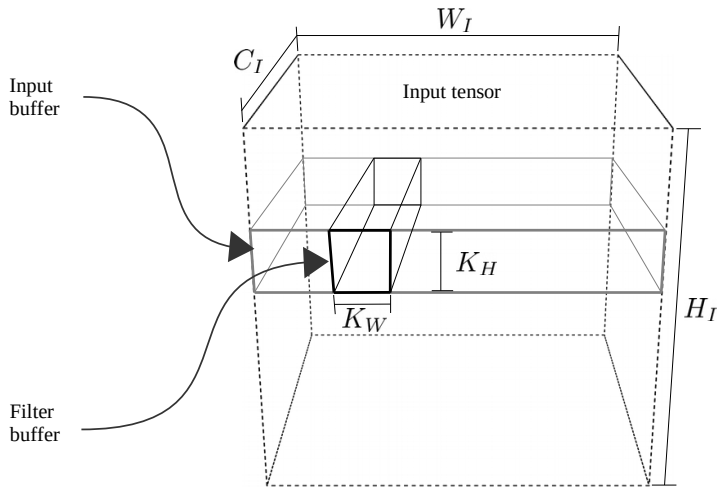


Figure: On-chip memory buffers

Dot-Product with Hybrid Floating-Point 6-bit

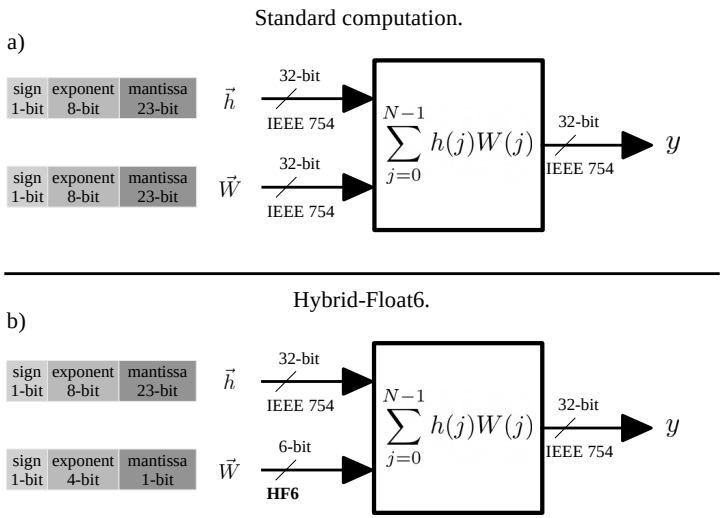


Figure: Dot-product

Dot-Product with Hybrid Floating-Point 6-bit

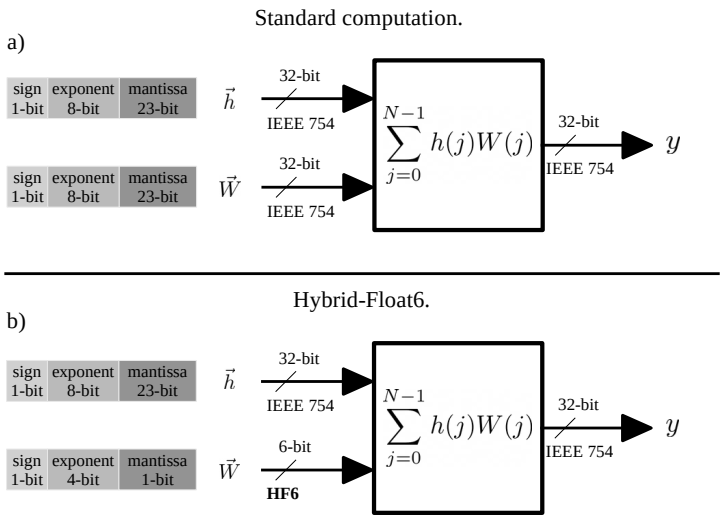


Figure: Dot-product

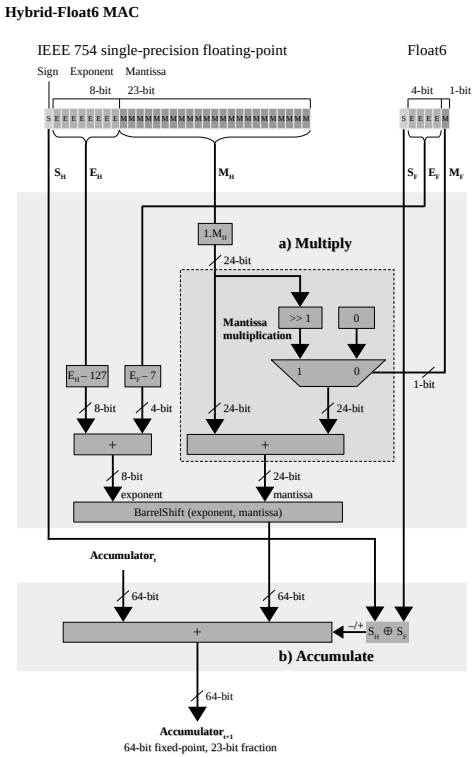


Figure: Multiply-accumulate design

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

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.

foreach *layer in* $MODEL$ **do**
  if *layer is* $Conv2D$ *or* $SeparableConv2D$ **then**
    $filter, bias \leftarrow GetWeights(layer)$
    foreach x *in* $filter$ *and* $bias$ **do**
      $sign \leftarrow GetSign(x)$
      $exp \leftarrow GetExponent(x)$
      $fullexp \leftarrow 2^{E_{size}-1} - 1$ `// Get full range value`
      $cman \leftarrow GetCustomMantissa(x, M_{size})$
      $leftman \leftarrow GetLeftoverMantissa(x, M_{size})$
      if $exp < -fullexp$ **then**
        $x \leftarrow 0$
      else
        if $exp > fullexp$ **then**
          $x \leftarrow (-1)^{sign} \cdot 2^{fullexp} \cdot (1 + (1 - 2^{-M_{size}}))$
        else
          if $2^{STDM_{size}-M_{size}-1} - 1 < leftman$ **then**
            $cman \leftarrow cman + 1$ `// Above halfway`
            if $2^{M_{size}} - 1 < cman$ **then**
              $cman \leftarrow 0$ `// Correct mantissa overflow`
              $exp \leftarrow exp + 1$
          $x \leftarrow (-1)^{sign} \cdot 2^{exp} \cdot (1 + cman \cdot 2^{-M_{size}})$
    $SetWeights(layer, filter, bias)$

Case Study: Applying Sensor Analytics to Structural Health Monitoring

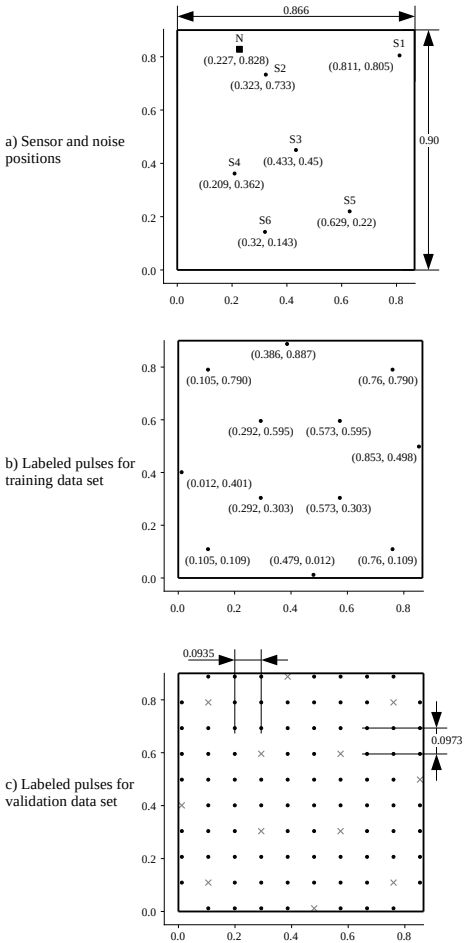


Figure: Structural health monitoring, all lengths are in meters (m)

Case Study: Applying Sensor Analytics to Structural Health Monitoring

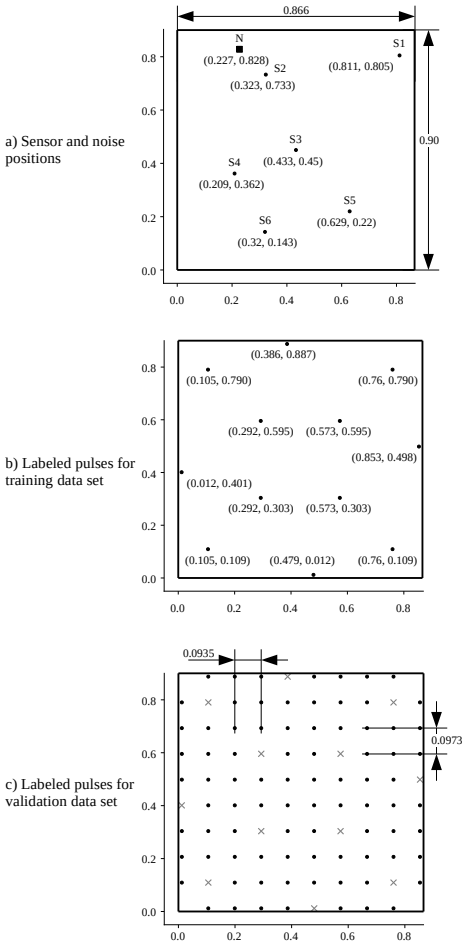


Figure: Structural health monitoring, all lengths are in meters (m)

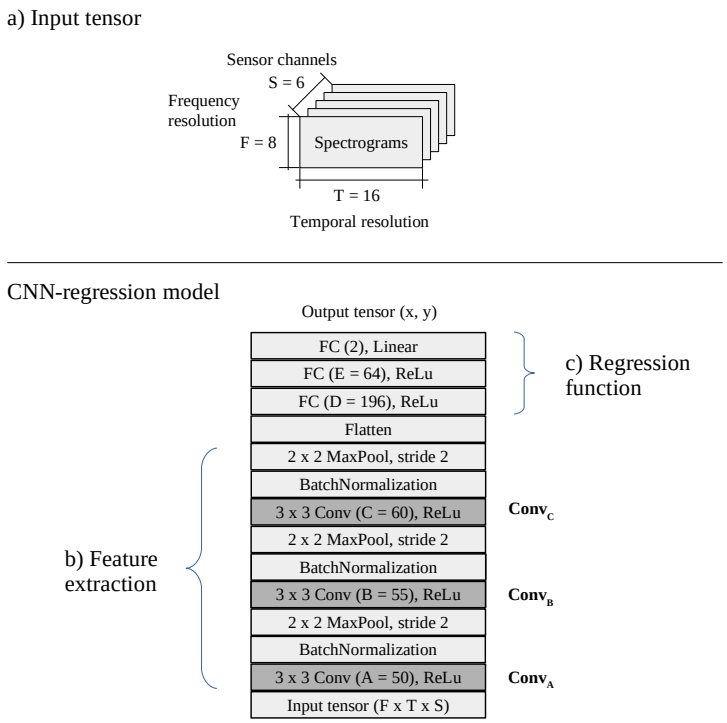


Figure: CNN-regression model for sensor analytics

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

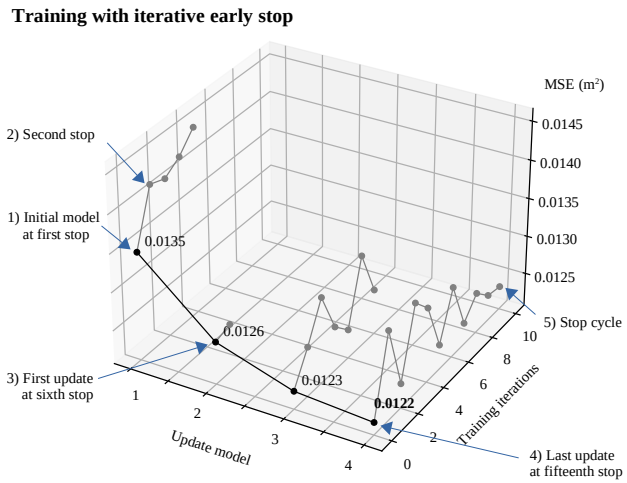
```
// Initial training and evaluation
Train(MODEL, D_train, D_val, N_E, B_size)
mse_i ← Evaluate(MODEL, D_val)
n_I ← 0
while n_I < N_I do
    // Iterative early stop cycle
    Train(MODEL, D_train, D_val, N_E, B_size)
    mse_v ← Evaluate(MODEL, D_val)
    if mse_v < mse_i then
        Update(MODEL)
        mse_i ← mse_v
    end
    else
        MODEL ← LoadPreviousWeights()
        n_I ← n_I + 1
    end
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
Train(MODEL, D_train, D_val, N_E, B_size)
mse_i ← Evaluate(MODEL, D_val)
n_I ← 0
while n_I < N_I do
    // Iterative early stop cycle
    Train(MODEL, D_train, D_val, N_E, B_size)
    mse_v ← Evaluate(MODEL, D_val)
    if mse_v < mse_i then
        Update(MODEL)
        mse_i ← mse_v
    end
    else
        MODEL ← LoadPreviousWeights()
        n_I ← n_I + 1
    end
end
```



Training

Algorithm 2: Training with iterative early stop cycle.

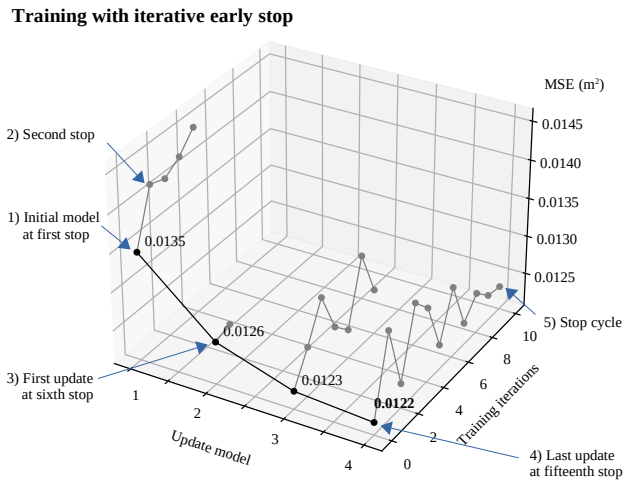
Input: $MODEL$ as the input model
Input: D_{train} as the training data set
Input: D_{val} as the validation data set
Input: N_I as the stop patience for iterative training cycle
Input: N_E as the early stop patience (epochs) for training
Input: B_{size} as the mini-batch size
Output: $MODEL$ as the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

```
// Initial training and evaluation
Train(MODEL, D_train, D_val, N_E, B_size)
mse_i ← Evaluate(MODEL, D_val)
n_I ← 0
while n_I < N_I do
    // Iterative early stop cycle
    Train(MODEL, D_train, D_val, N_E, B_size)
    mse_v ← Evaluate(MODEL, D_val)
    if mse_v < mse_i then
        Update(MODEL)
        mse_i ← mse_v
    end
    else
        MODEL ← LoadPreviousWeights()
        n_I ← n_I + 1
    end
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
MODEL ← QuantizeTraining(MODEL, E_size, M_size)
if 1 < epoch then
    // Update model after first epoch
    mse_v ← Evaluate(MODEL, D_val)
    if mse_v < mse_i then
        Update(MODEL)
        mse_i ← mse_v
    end
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

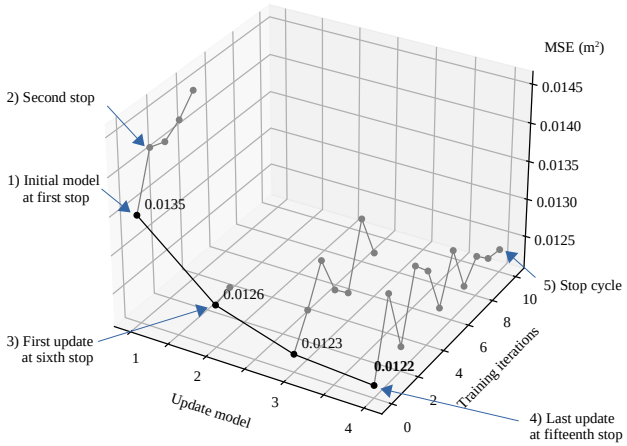
```
// Initial training and evaluation
Train(MODEL, D_train, D_val, N_E, B_size)
mse_i ← Evaluate(MODEL, D_val)
n_I ← 0
while n_I < N_I do
    // Iterative early stop cycle
    Train(MODEL, D_train, D_val, N_E, B_size)
    mse_v ← Evaluate(MODEL, D_val)
    if mse_v < mse_i then
        Update(MODEL)
        mse_i ← mse_v
    end
    else
        MODEL ← LoadPreviousWeights()
        n_I ← n_I + 1
    end
end
```

Algorithm 3: OnMiniBatchUpdate_Callback.

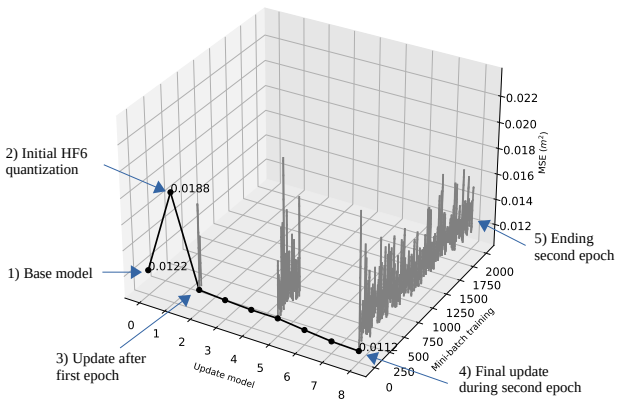
Input: $MODEL$ as the full-precision input model
Input: E_{size} as the target exponent bits size
Input: M_{size} as the target mantissa bits size
Input: D_{train} as the training data set
Input: D_{val} as the validation data set
Input: N_{ep} as the number of epochs
Input: B_{size} as the mini-batch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

```
// Quantize
MODEL ← QuantizeTraining(MODEL, E_size, M_size)
if 1 < epoch then
    // Update model after first epoch
    mse_v ← Evaluate(MODEL, D_val)
    if mse_v < mse_i then
        Update(MODEL)
        mse_i ← mse_v
    end
end
```

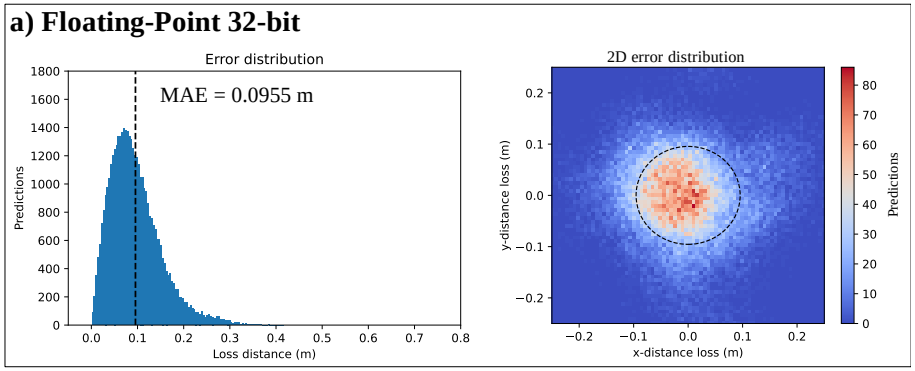
Training with iterative early stop



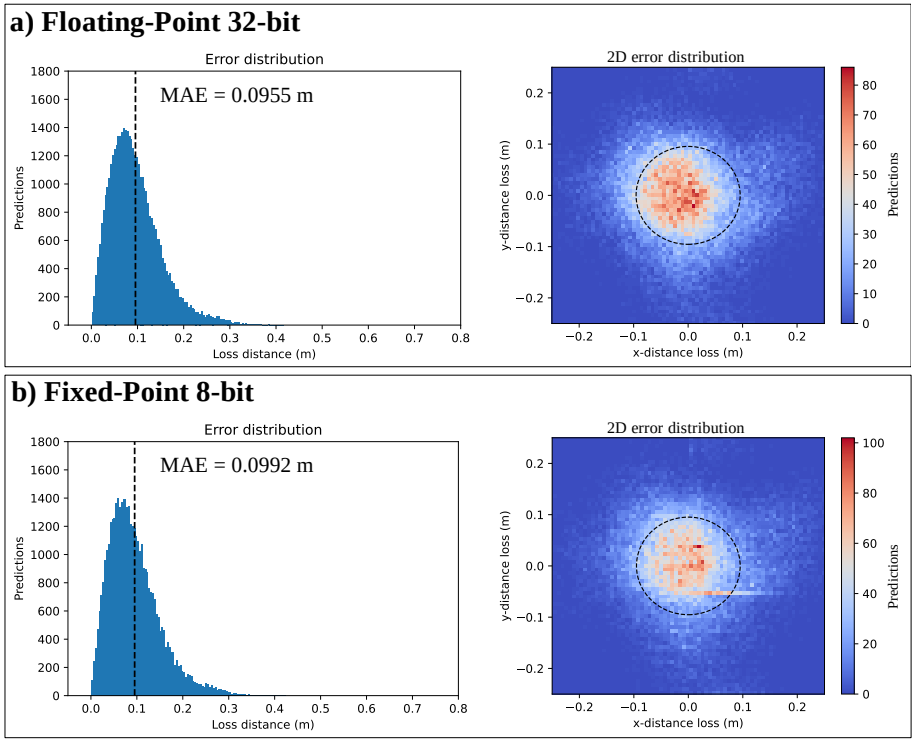
Quantization-aware training



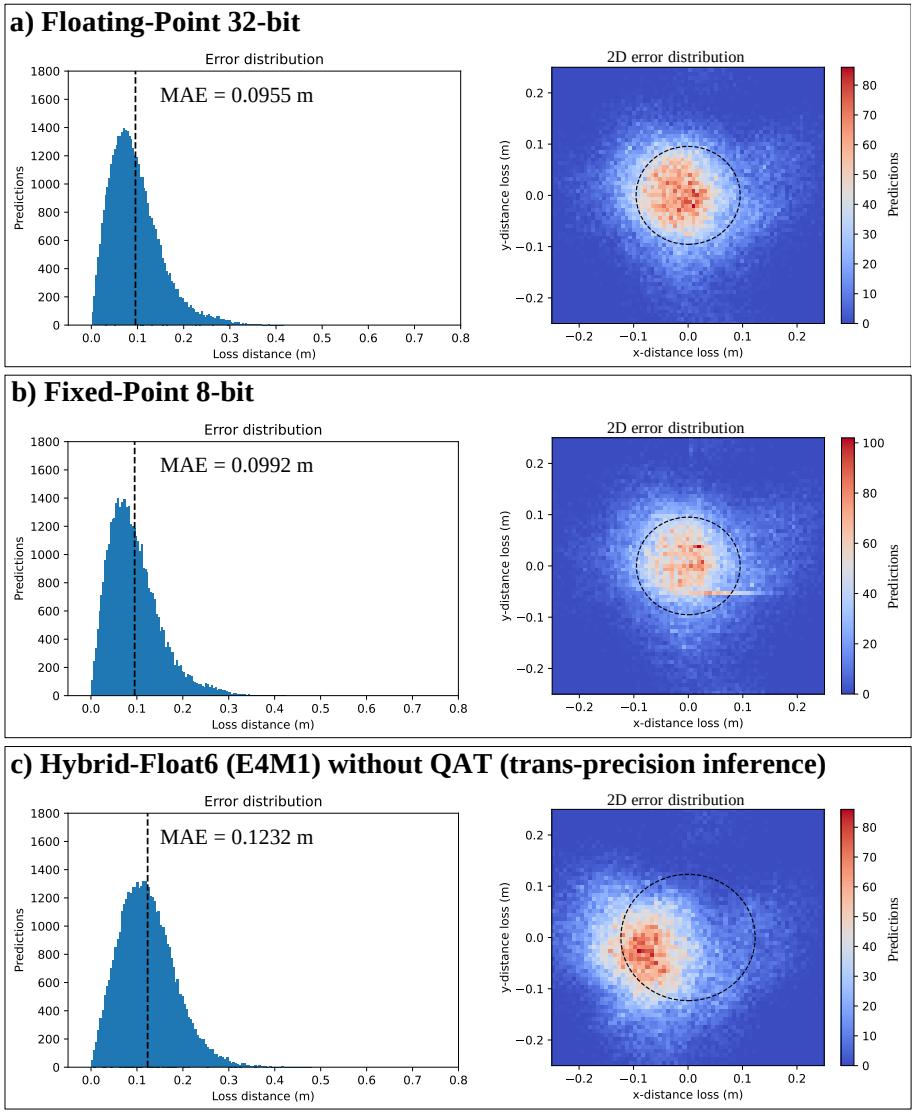
Quantization Impact: Error Histograms in Position Prediction



Quantization Impact: Error Histograms in Position Prediction

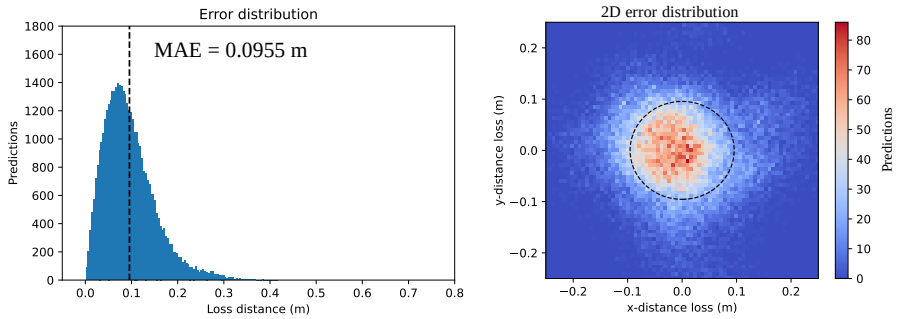


Quantization Impact: Error Histograms in Position Prediction

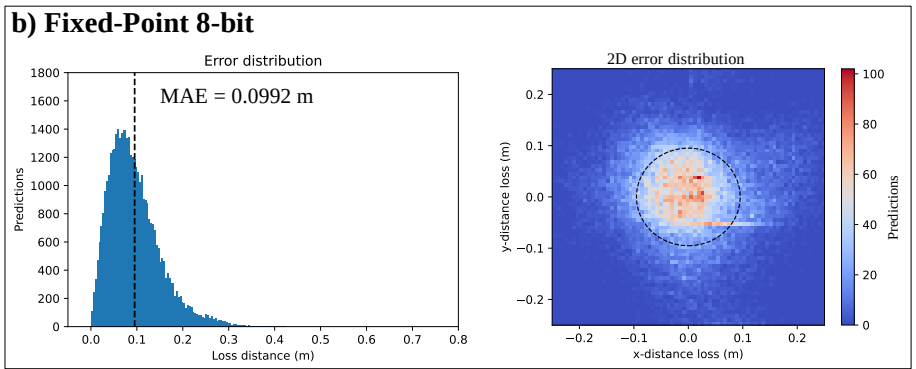


Quantization Impact: Error Histograms in Position Prediction

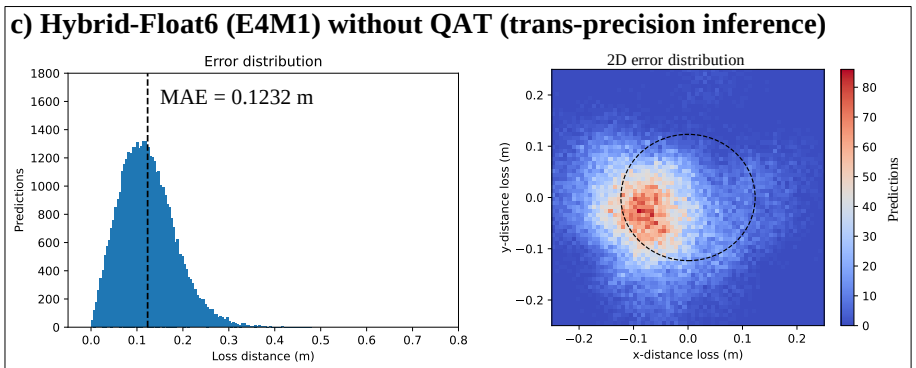
a) Floating-Point 32-bit



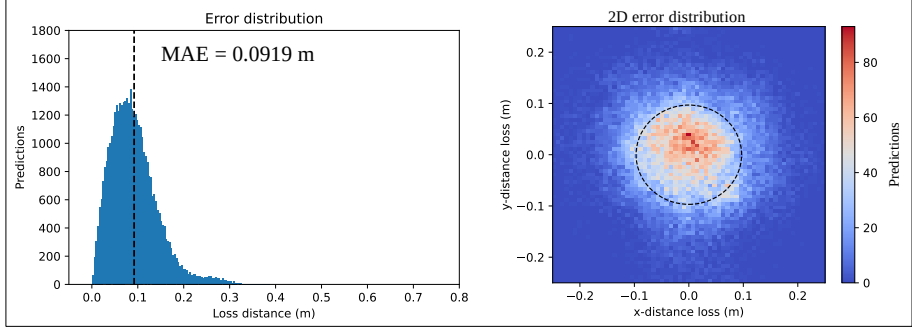
b) Fixed-Point 8-bit



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

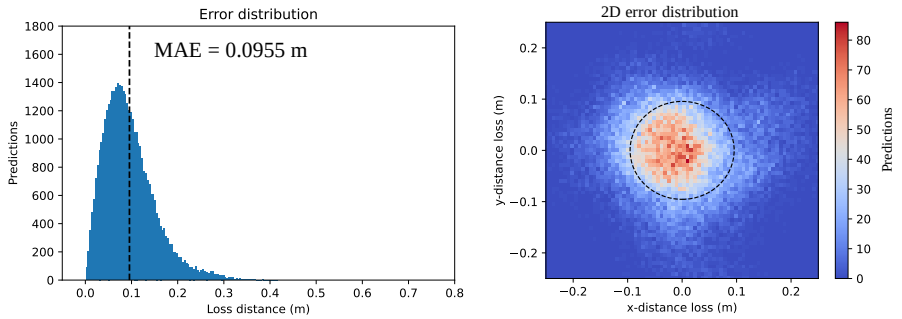


d) Hybrid-Float6 (E4M1) with QAT

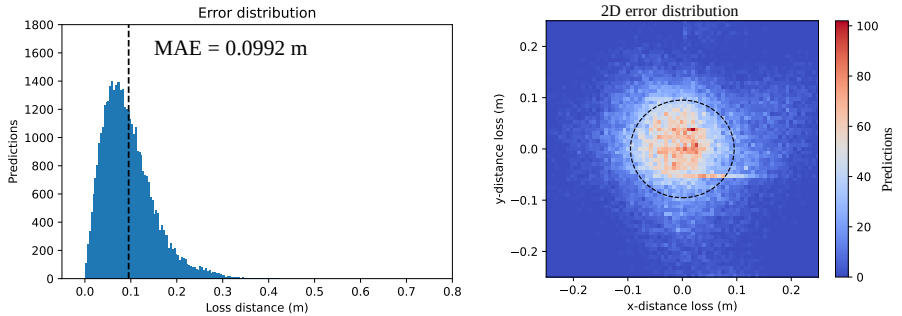


Quantization Impact: Error Histograms in Position Prediction

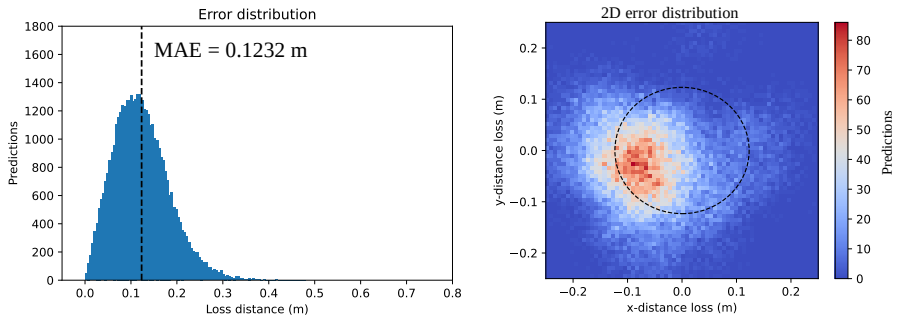
a) Floating-Point 32-bit



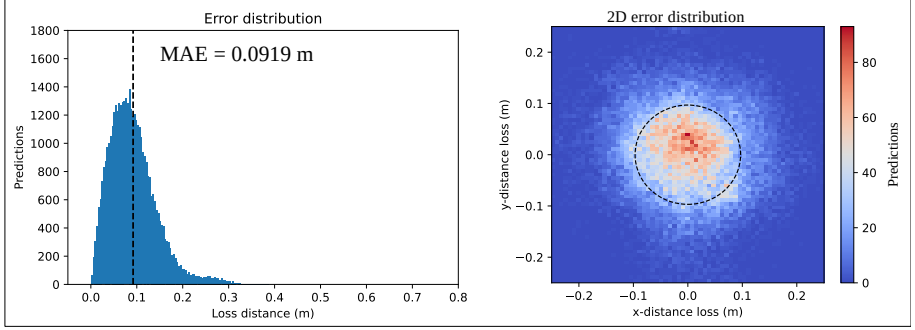
b) Fixed-Point 8-bit



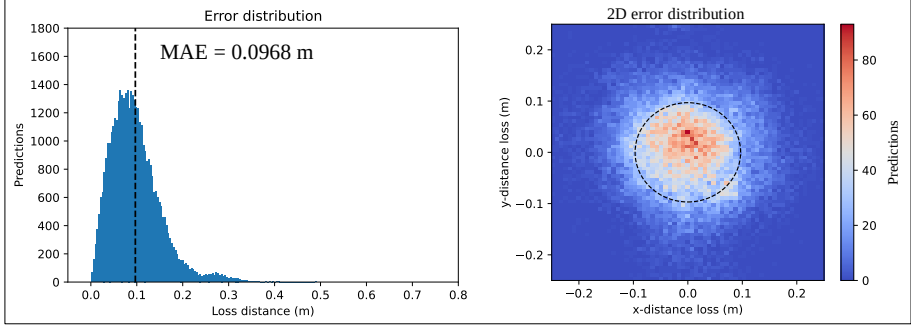
c) Hybrid-Float6 (E4M1) without QAT (trans-precision inference)



d) Hybrid-Float6 (E4M1) with QAT

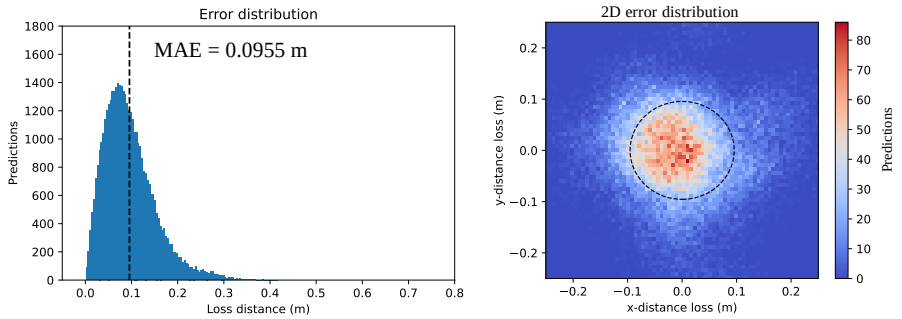


e) Hybrid-Logarithmic 6-bit (E5M0) with QAT

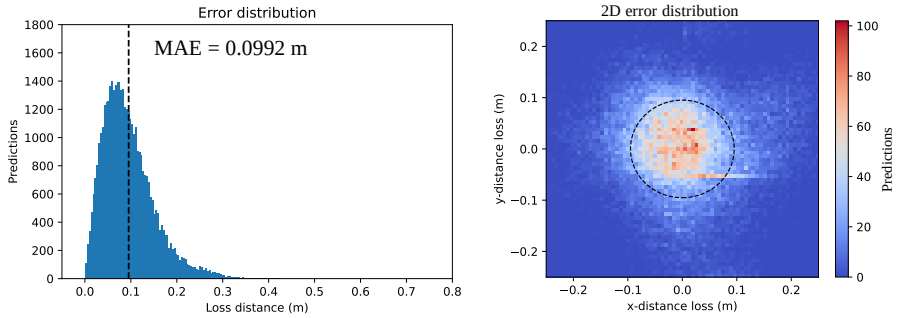


Quantization Impact: Error Histograms in Position Prediction

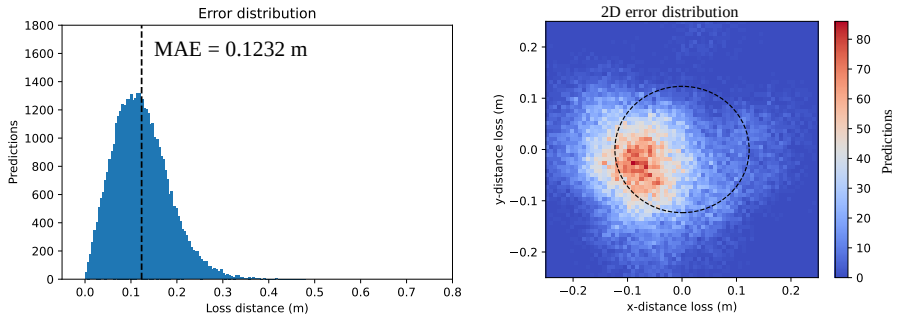
a) Floating-Point 32-bit



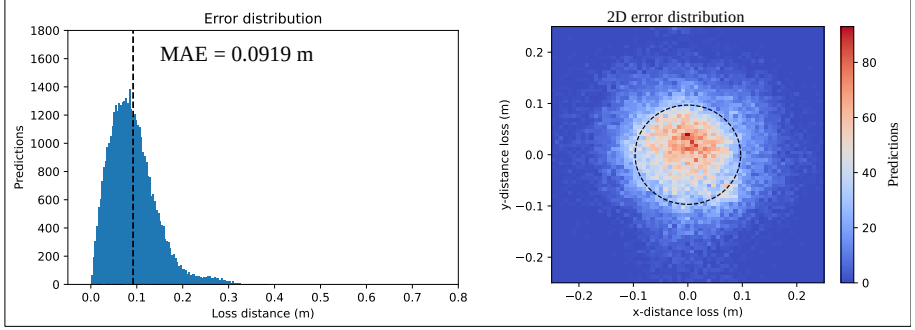
b) Fixed-Point 8-bit



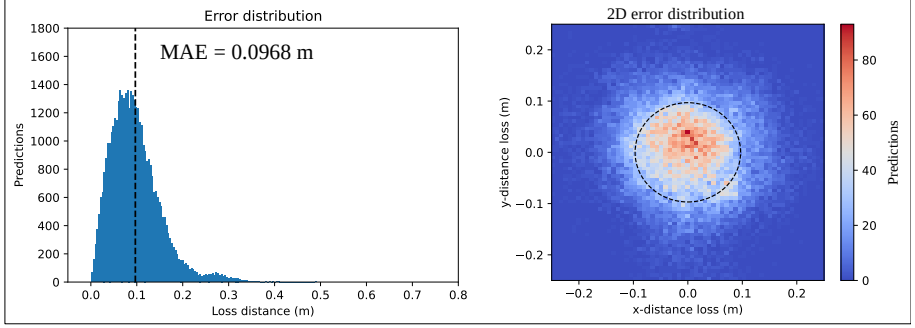
c) Hybrid-Float6 (E4M1) without QAT (trans-precision inference)



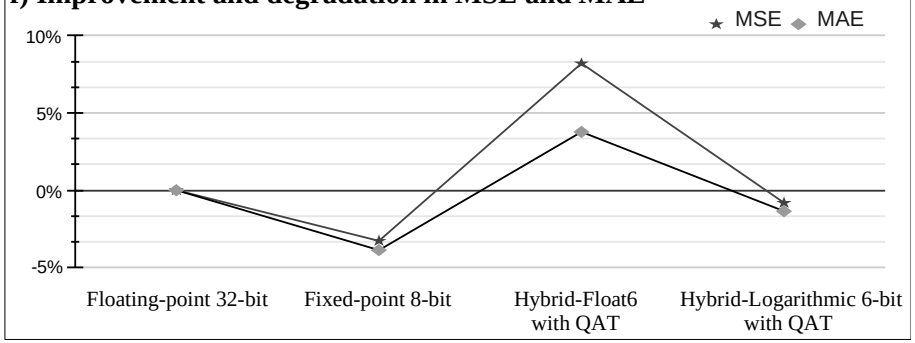
d) Hybrid-Float6 (E4M1) with QAT



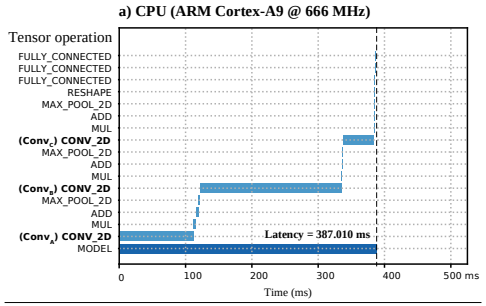
e) Hybrid-Logarithmic 6-bit (E5M0) with QAT



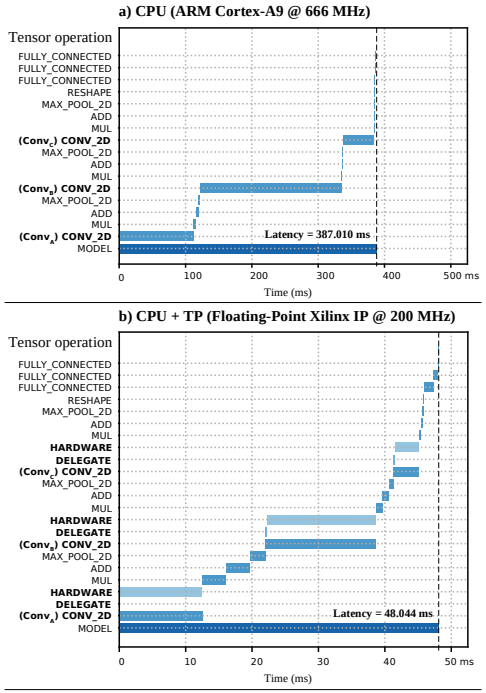
f) Improvement and degradation in MSE and MAE



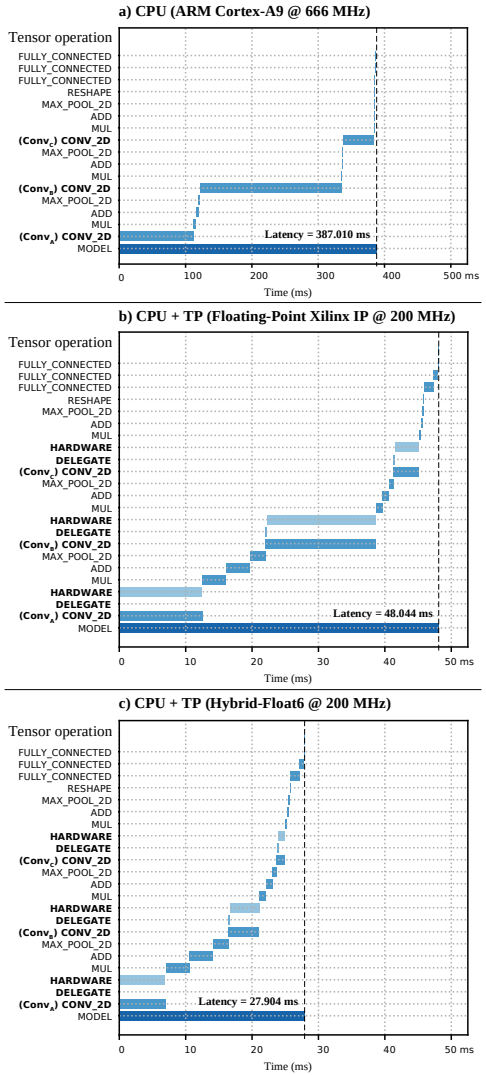
Evaluating Floating-Point Configurations: Speed, Power, and Hardware Utilization



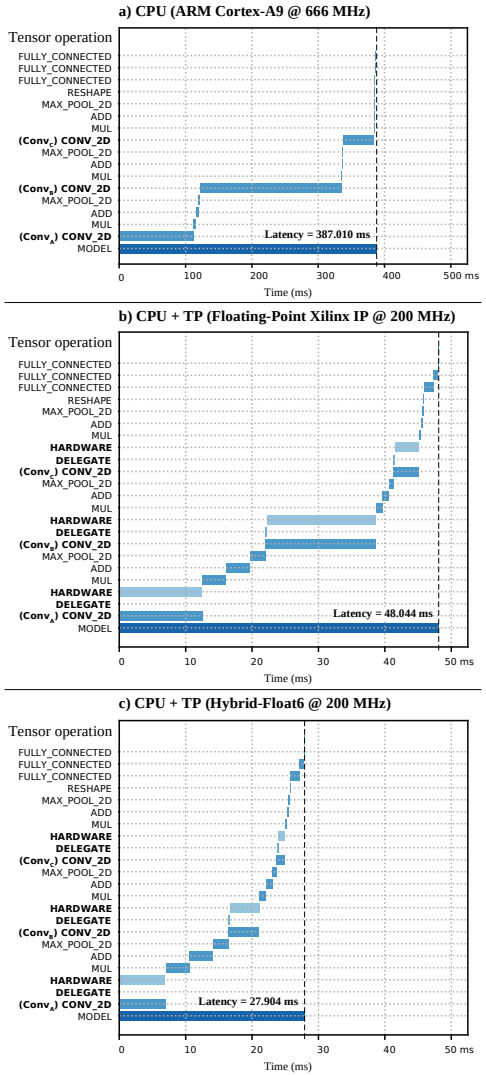
Evaluating Floating-Point Configurations: Speed, Power, and Hardware Utilization

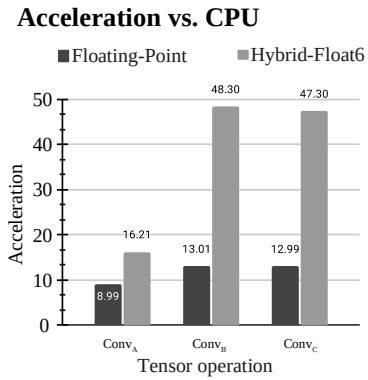


Evaluating Floating-Point Configurations: Speed, Power, and Hardware Utilization

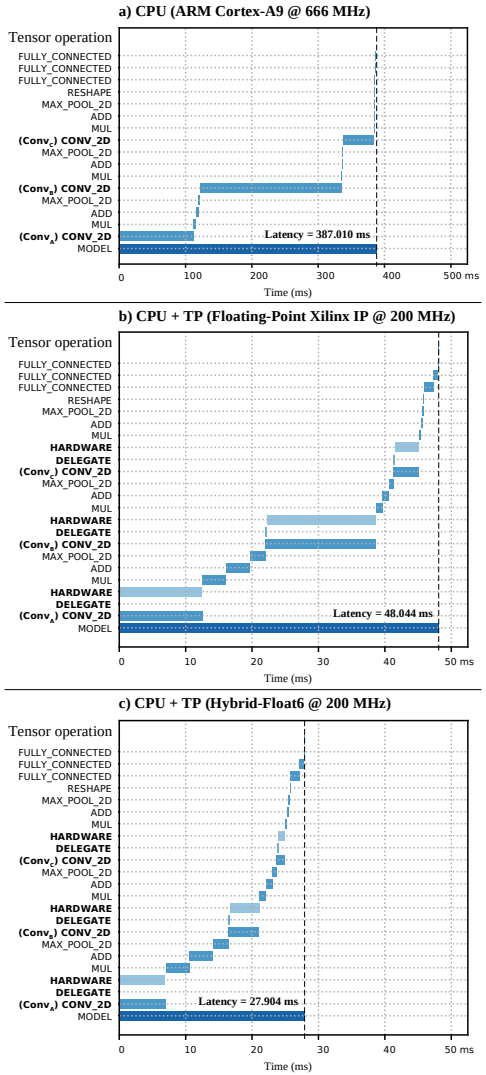


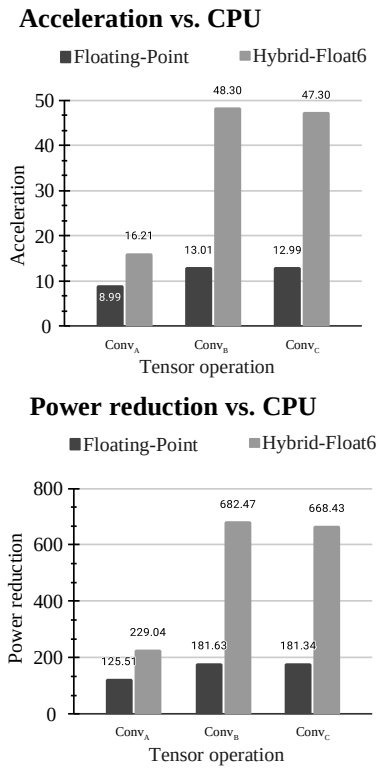
Evaluating Floating-Point Configurations: Speed, Power, and Hardware Utilization



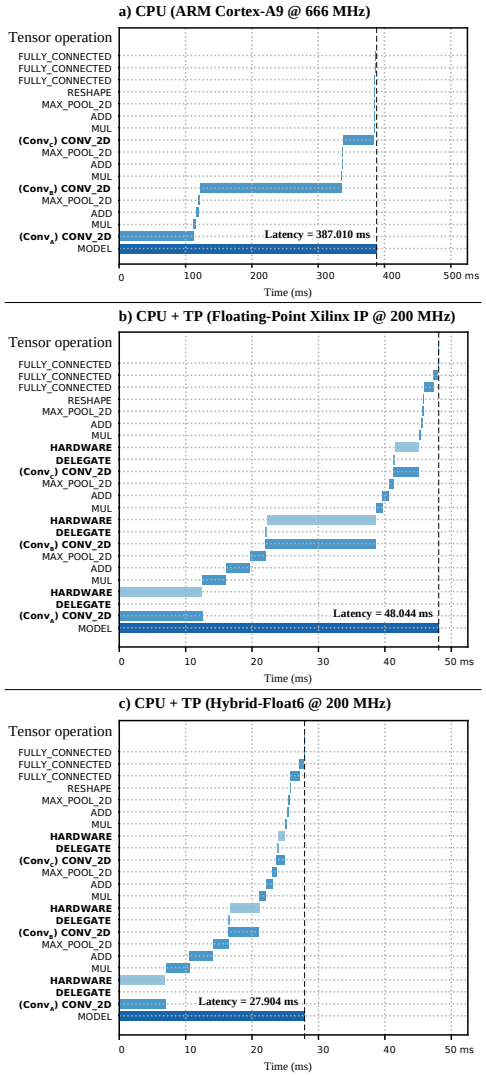


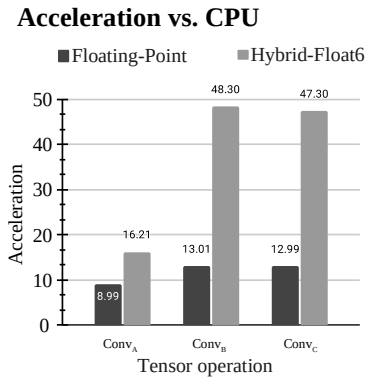
Evaluating Floating-Point Configurations: Speed, Power, and Hardware Utilization

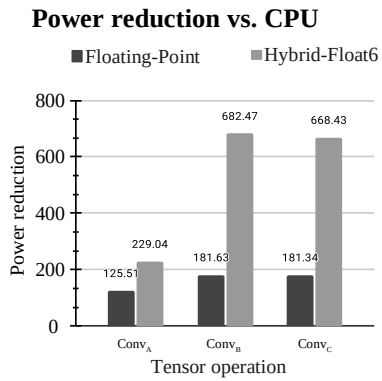


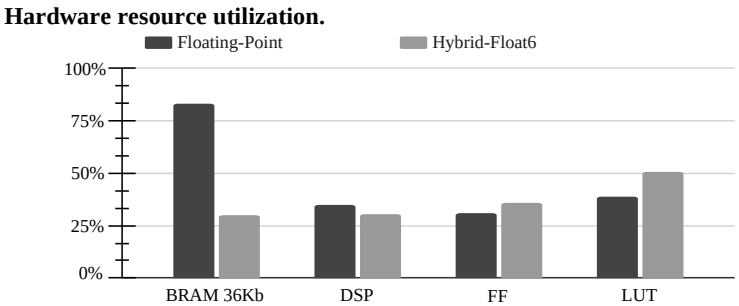


Evaluating Floating-Point Configurations: Speed, Power, and Hardware Utilization









Comparison with Related Work

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

Platform	Chunsheng *et al.* [1]	Chen *et al.* [2]	BFP [3]	Paolo *et al.* [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.

Conclusions and Future Research

Conclusions

- It was presented a design methodology for low-power Spike-by-Spike neural network accelerators with **hybrid 8-bit floating-point and 4-bit logarithm computation**

Conclusions and Future Research

Conclusions

- It was presented a design methodology for low-power Spike-by-Spike neural network accelerators with **hybrid 8-bit floating-point and 4-bit logarithm computation**
- It was presented a design methodology for low-power custom floating-point CNN accelerators showcasing the potential of **hybrid 6-bit quantization**

Conclusions and Future Research

Conclusions

- It was presented a design methodology for low-power Spike-by-Spike neural network accelerators with **hybrid 8-bit floating-point and 4-bit logarithm computation**
- It was presented a design methodology for low-power custom floating-point CNN accelerators showcasing the potential of **hybrid 6-bit quantization**

Conclusions and Future Research

Conclusions

- It was presented a design methodology for low-power Spike-by-Spike neural network accelerators with **hybrid 8-bit floating-point and 4-bit logarithm computation**
- It was presented a design methodology for low-power custom floating-point CNN accelerators showcasing the potential of **hybrid 6-bit quantization**

Future Research

- Low-power neural network accelerators for **on-device training** with hybrid custom floating-point computation

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.