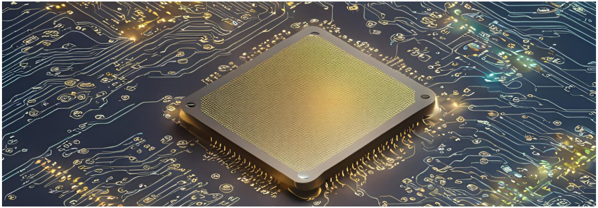


Low-Power Neural Network Accelerators: Advancements in Custom Floating-Point Techniques

Yarib Nevarez

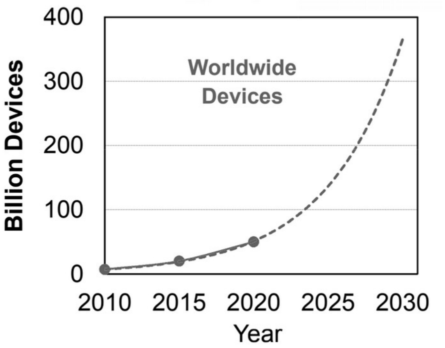
Universität Bremen

May 22, 2024

Expansion of IoT and TinyML: Requirements, Accelerators, and Challenges

Internet-of-Things (IoT) in Smart Cities and Industry 4.0

Any device	Anybody	Anywhere	Any business	Any network	Anytime



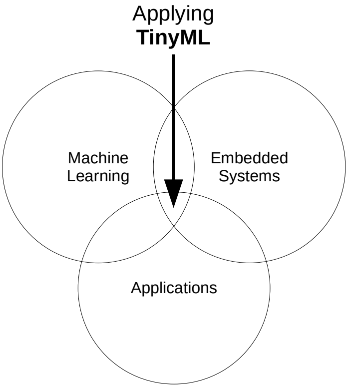
Loh, Kou-Hung Lawrence. "1.2 Fertilizing AIoT from roots to leaves." In 2020 IEEE International Solid-State Circuits Conference-(ISSCC), pp. 15-21. IEEE, 2020.

Introduction

Expansion of IoT and TinyML: Requirements, Accelerators, and Challenges

Internet-of-Things (IoT) in Smart Cities and Industry 4.0

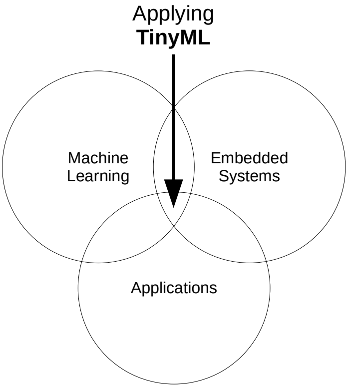
Any device	Anybody	Anywhere	Any business	Any network	Anytime



Expansion of IoT and TinyML: Requirements, Accelerators, and Challenges

Internet-of-Things (IoT) in Smart Cities and Industry 4.0

Neural network accelerators



Expansion of IoT and TinyML: Requirements, Accelerators, and Challenges

Internet-of-Things (IoT) in Smart Cities and Industry 4.0

Neural network accelerators

Aspects for long-term sustainability:

- Energy and resource efficiency
- Quality preservation
- Application versatility
- Platform compatibility
- On-device training

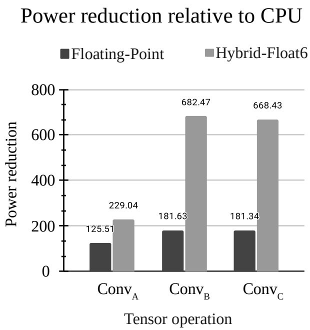
Expansion of IoT and TinyML: Requirements, Accelerators, and Challenges

Internet-of-Things (IoT) in Smart Cities and Industry 4.0

Neural network accelerators

Aspects for long-term sustainability:

- **Energy and resource efficiency**
- Quality preservation
- Application versatility
- Platform compatibility
- On-device training



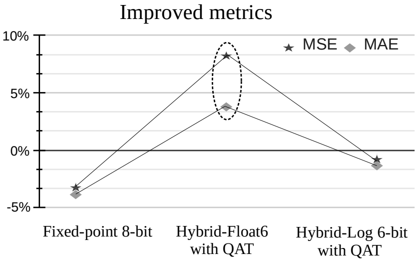
Expansion of IoT and TinyML: Requirements, Accelerators, and Challenges

Internet-of-Things (IoT) in Smart Cities and Industry 4.0

Neural network accelerators

Aspects for long-term sustainability:

- Energy and resource efficiency
- **Quality preservation**
- Application versatility
- Platform compatibility
- On-device training

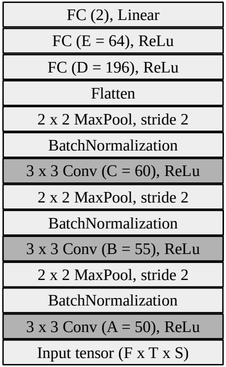


Expansion of IoT and TinyML: Requirements, Accelerators, and Challenges

Internet-of-Things (IoT) in Smart Cities and Industry 4.0

Neural network accelerators

Model versatility



Aspects for long-term sustainability:

Energy and resource efficiency

Quality preservation

Application versatility

Platform compatibility

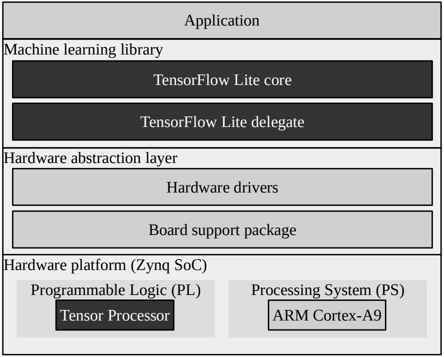
On-device training

Expansion of IoT and TinyML: Requirements, Accelerators, and Challenges

Internet-of-Things (IoT) in Smart Cities and Industry 4.0

Neural network accelerators

HW/SW co-design framework



Aspects for long-term sustainability:

- Energy and resource efficiency
- Quality preservation
- Application versatility
- **Platform compatibility**
- On-device training

Expansion of IoT and TinyML: Requirements, Accelerators, and Challenges

Internet-of-Things (IoT) in Smart Cities and Industry 4.0

Neural network accelerators

Aspects for long-term sustainability:

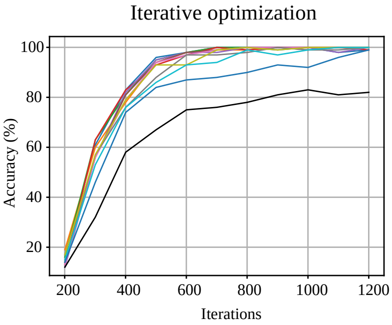
Energy and resource efficiency

Quality preservation

Application versatility

Platform compatibility

On-device training



Expansion of IoT and TinyML: Requirements, Accelerators, and Challenges

Internet-of-Things (IoT) in Smart Cities and Industry 4.0

Neural network accelerators

Aspects for long-term sustainability:

Energy and resource efficiency

Quality preservation

Application versatility

Platform compatibility

On-device training

Expansion of IoT and TinyML: Requirements, Accelerators, and Challenges

Internet-of-Things (IoT) in Smart Cities and Industry 4.0

Neural network accelerators

Aspects for long-term sustainability:

- **Energy and resource efficiency**
- **Quality preservation**
- **Application versatility**
- **Platform compatibility**
- **On-device training**

Current state-of-the-art methods:

Expansion of IoT and TinyML: Requirements, Accelerators, and Challenges

Internet-of-Things (IoT) in Smart Cities and Industry 4.0

Neural network accelerators

Aspects for long-term sustainability:

- **Energy and resource efficiency**
- **Quality preservation**
- **Application versatility**
- **Platform compatibility**
- **On-device training**

Current state-of-the-art methods:

Extreme quantization
Fails to adequately meet fundamental aspects, particularly in complex problems and mission-critical applications.

Expansion of IoT and TinyML: Requirements, Accelerators, and Challenges

Internet-of-Things (IoT) in Smart Cities and Industry 4.0

Neural network accelerators

Aspects for long-term sustainability:

- **Energy and resource efficiency**
- **Quality preservation**
- **Application versatility**
- **Platform compatibility**
- **On-device training**

Current state-of-the-art methods:

Extreme quantization
Fails to adequately meet fundamental aspects, particularly in complex problems and mission-critical applications.

Fixed precision
Fails to adequately adapt to the ongoing technological shift towards on-device training.

Goal and Objectives

- **Goal**: To establish a sustainable, efficient, and universally applicable neural network acceleration technique that supports inference and facilitates on-device training in extreme edge devices, ensuring future-proof and broad applicability. This approach integrates hybrid custom floating-point (FP) computation.

- **Goal**: To establish a sustainable, efficient, and universally applicable neural network acceleration technique that supports inference and facilitates on-device training in extreme edge devices, ensuring future-proof and broad applicability. This approach integrates hybrid custom floating-point (FP) computation.

- **Objectives**:

- **Goal**: To establish a sustainable, efficient, and universally applicable neural network acceleration technique that supports inference and facilitates on-device training in extreme edge devices, ensuring future-proof and broad applicability. This approach integrates hybrid custom floating-point (FP) computation.

- **Objectives**:
  - Investigate custom FP computation

Goal and Objectives

- **Goal**: To establish a sustainable, efficient, and universally applicable neural network acceleration technique that supports inference and facilitates on-device training in extreme edge devices, ensuring future-proof and broad applicability. This approach integrates hybrid custom floating-point (FP) computation.

- **Objectives**:
  - Investigate custom FP computation
  - Conduct performance evaluation

Goal and Objectives

- **Goal**: To establish a sustainable, efficient, and universally applicable neural network acceleration technique that supports inference and facilitates on-device training in extreme edge devices, ensuring future-proof and broad applicability. This approach integrates hybrid custom floating-point (FP) computation.

- **Objectives**:
  - Investigate custom FP computation
  - Conduct performance evaluation
  - Analyze impact

Goal and Objectives

- **Goal**: To establish a sustainable, efficient, and universally applicable neural network acceleration technique that supports inference and facilitates on-device training in extreme edge devices, ensuring future-proof and broad applicability. This approach integrates hybrid custom floating-point (FP) computation.

- **Objectives**:
  - Investigate custom FP computation
  - Conduct performance evaluation
  - Analyze impact
  - Ensure cross-platform compatibility

Outline

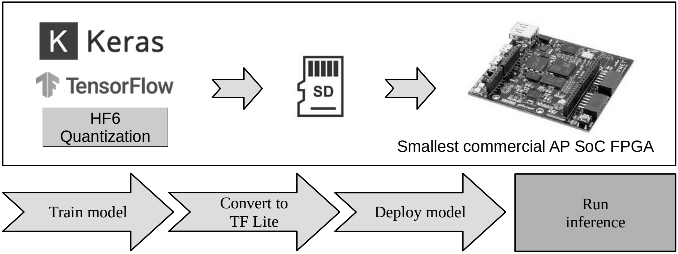
Methodology

Trans-Precision Neural Network Deployment for Low-Power Embedded Systems

The methodology efficiently deploys and accelerates floating-point neural networks on embedded systems, optimizing performance, energy consumption, and hardware utilization.

Characteristics:

- Streamlined hardware architecture
- Custom floating-point arithmetic
- Neural network execution without quantization processes

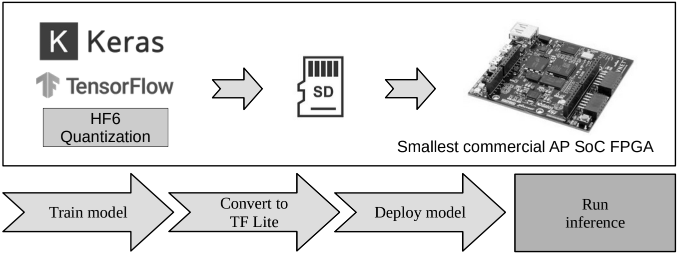


Trans-Precision Neural Network Deployment for Low-Power Embedded Systems

The methodology efficiently deploys and accelerates floating-point neural networks on embedded systems, optimizing performance, energy consumption, and hardware utilization.

Characteristics:

- Streamlined hardware architecture
- Custom floating-point arithmetic
- Neural network execution without quantization processes

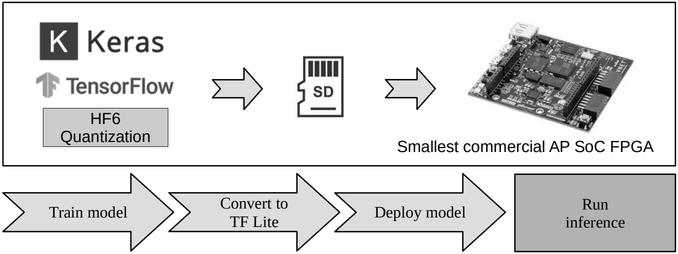


Trans-Precision Neural Network Deployment for Low-Power Embedded Systems

The methodology efficiently deploys and accelerates floating-point neural networks on embedded systems, optimizing performance, energy consumption, and hardware utilization.

Characteristics:

- Streamlined hardware architecture
- Custom floating-point arithmetic
- Neural network execution without quantization processes

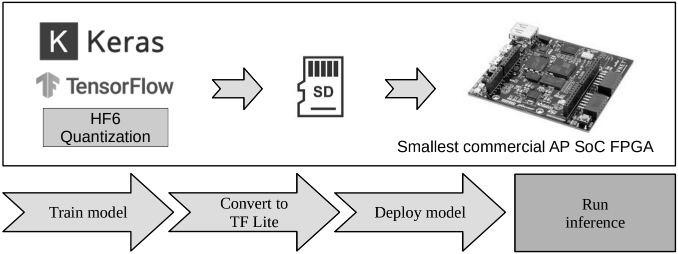


Trans-Precision Neural Network Deployment for Low-Power Embedded Systems

The methodology efficiently deploys and accelerates floating-point neural networks on embedded systems, optimizing performance, energy consumption, and hardware utilization.

Characteristics:

- Streamlined hardware architecture
- Custom floating-point arithmetic
- Neural network execution without quantization processes



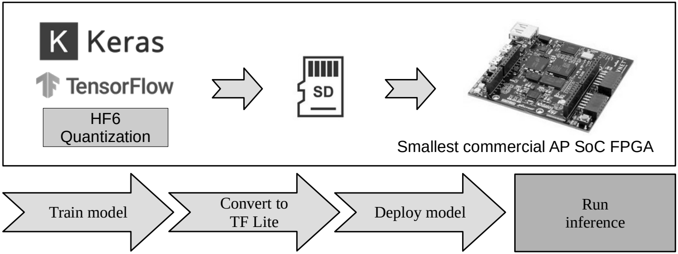
Methodology

Trans-Precision Neural Network Deployment for Low-Power Embedded Systems

The methodology efficiently deploys and accelerates floating-point neural networks on embedded systems, optimizing performance, energy consumption, and hardware utilization.

Characteristics:

- Streamlined hardware architecture
- Custom floating-point arithmetic
- Neural network execution without quantization processes



Trans-Precision Neural Network Deployment for Low-Power Embedded Systems

The methodology efficiently deploys and accelerates floating-point neural networks on embedded systems, optimizing performance, energy consumption, and hardware utilization.

Abstraction levels:

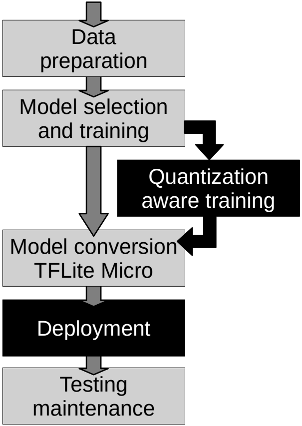
1. Model deployment
2. System infrastructure
3. Streamlined acceleration
4. Optimized processing

Trans-Precision Neural Network Deployment for Low-Power Embedded Systems

The methodology efficiently deploys and accelerates floating-point neural networks on embedded systems, optimizing performance, energy consumption, and hardware utilization.

Abstraction levels:

1. **Model deployment**
2. System infrastructure
3. Streamlined acceleration
4. Optimized processing

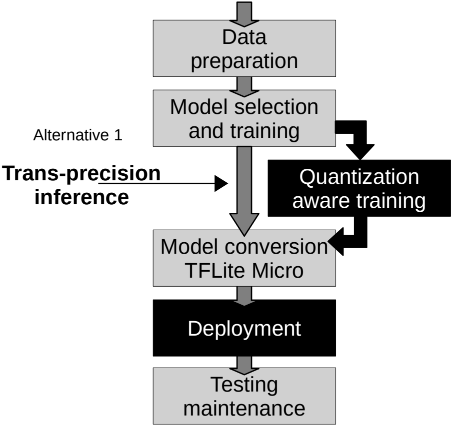


Trans-Precision Neural Network Deployment for Low-Power Embedded Systems

The methodology efficiently deploys and accelerates floating-point neural networks on embedded systems, optimizing performance, energy consumption, and hardware utilization.

Abstraction levels:

1. **Model deployment**
2. System infrastructure
3. Streamlined acceleration
4. Optimized processing

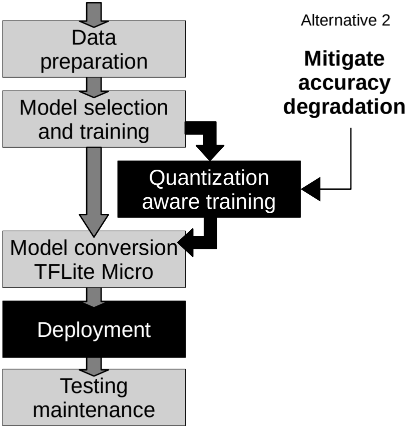


Trans-Precision Neural Network Deployment for Low-Power Embedded Systems

The methodology efficiently deploys and accelerates floating-point neural networks on embedded systems, optimizing performance, energy consumption, and hardware utilization.

Abstraction levels:

1. **Model deployment**
2. System infrastructure
3. Streamlined acceleration
4. Optimized processing

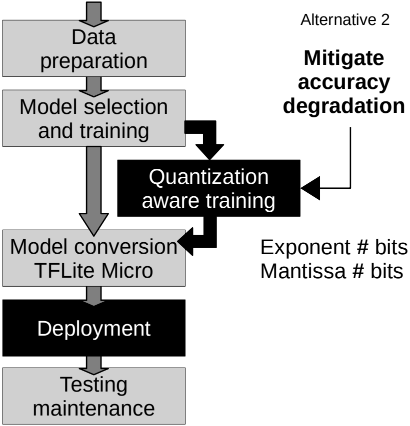


Trans-Precision Neural Network Deployment for Low-Power Embedded Systems

The methodology efficiently deploys and accelerates floating-point neural networks on embedded systems, optimizing performance, energy consumption, and hardware utilization.

Abstraction levels:

1. **Model deployment**
2. System infrastructure
3. Streamlined acceleration
4. Optimized processing



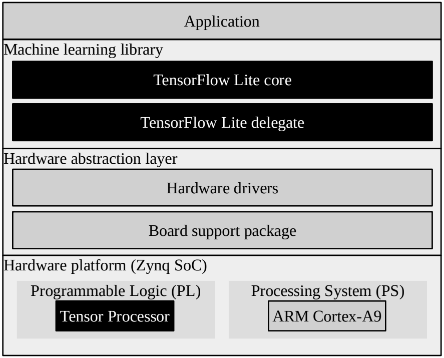
Trans-Precision Neural Network Deployment for Low-Power Embedded Systems

The methodology efficiently deploys and accelerates floating-point neural networks on embedded systems, optimizing performance, energy consumption, and hardware utilization.

Abstraction levels:

1. Model deployment
2. **System infrastructure**
3. Streamlined acceleration
4. Optimized processing

HW/SW co-design framework



Methodology

Trans-Precision Neural Network Deployment for Low-Power Embedded Systems

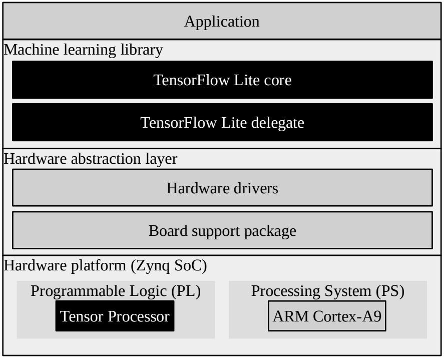
The methodology efficiently deploys and accelerates floating-point neural networks on embedded systems, optimizing performance, energy consumption, and hardware utilization.

Abstraction levels:

1. Model deployment
2. **System infrastructure**
3. Streamlined acceleration
4. Optimized processing

Industry standard framework

HW/SW co-design framework

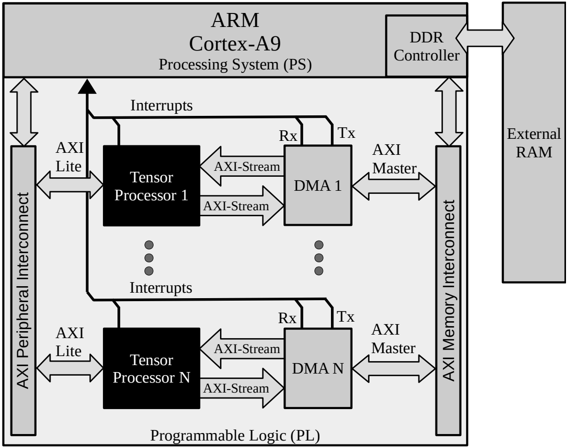


Trans-Precision Neural Network Deployment for Low-Power Embedded Systems

The methodology efficiently deploys and accelerates floating-point neural networks on embedded systems, optimizing performance, energy consumption, and hardware utilization.

Abstraction levels:

1. Model deployment
2. System infrastructure
3. **Streamlined acceleration**
4. Optimized processing

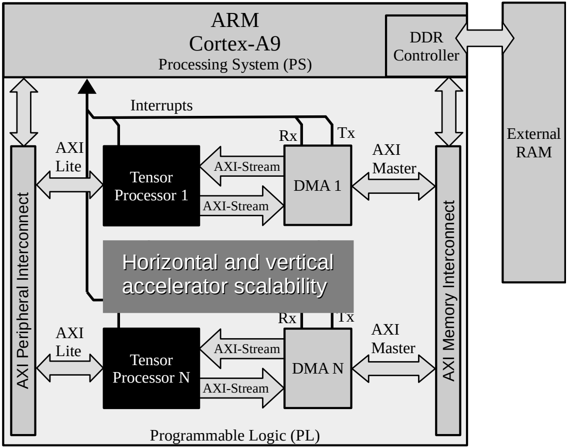


Trans-Precision Neural Network Deployment for Low-Power Embedded Systems

The methodology efficiently deploys and accelerates floating-point neural networks on embedded systems, optimizing performance, energy consumption, and hardware utilization.

Abstraction levels:

1. Model deployment
2. System infrastructure
3. **Streamlined acceleration**
4. Optimized processing

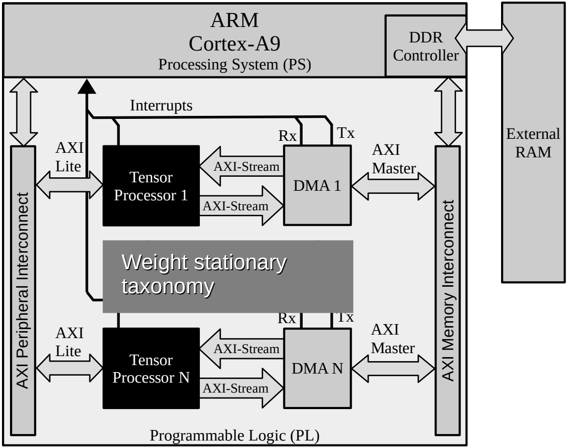


Trans-Precision Neural Network Deployment for Low-Power Embedded Systems

The methodology efficiently deploys and accelerates floating-point neural networks on embedded systems, optimizing performance, energy consumption, and hardware utilization.

Abstraction levels:

1. Model deployment
2. System infrastructure
3. **Streamlined acceleration**
4. Optimized processing



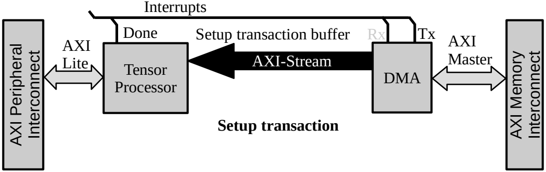
Trans-Precision Neural Network Deployment for Low-Power Embedded Systems

The methodology efficiently deploys and accelerates floating-point neural networks on embedded systems, optimizing performance, energy consumption, and hardware utilization.

Abstraction levels:

1. Model deployment
2. System infrastructure
3. **Streamlined acceleration**
4. Optimized processing

Configuration mode

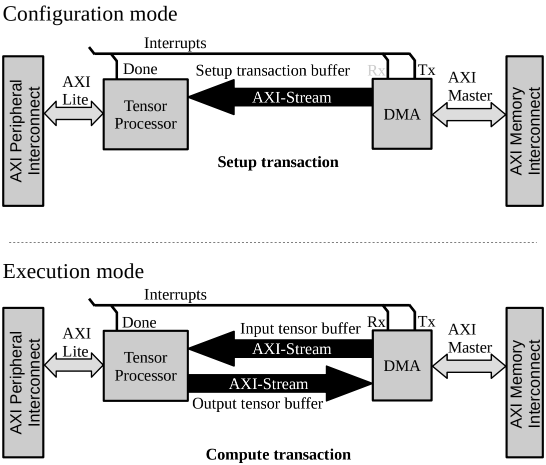


Trans-Precision Neural Network Deployment for Low-Power Embedded Systems

The methodology efficiently deploys and accelerates floating-point neural networks on embedded systems, optimizing performance, energy consumption, and hardware utilization.

Abstraction levels:

1. Model deployment
2. System infrastructure
3. **Streamlined acceleration**
4. Optimized processing

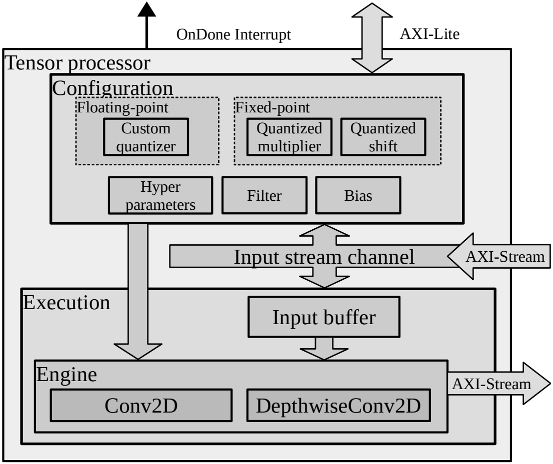


Trans-Precision Neural Network Deployment for Low-Power Embedded Systems

The methodology efficiently deploys and accelerates floating-point neural networks on embedded systems, optimizing performance, energy consumption, and hardware utilization.

Abstraction levels:

1. Model deployment
2. System infrastructure
3. **Streamlined acceleration**
4. Optimized processing

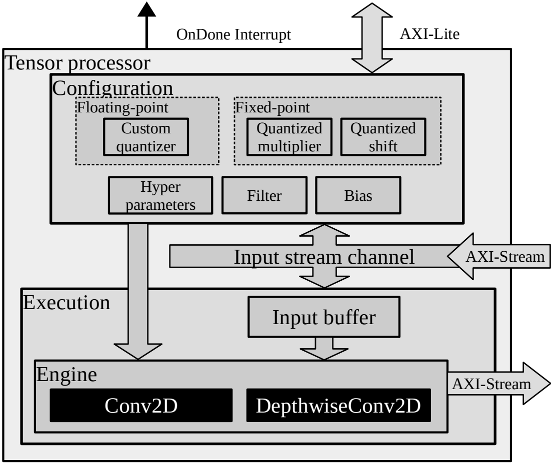


Trans-Precision Neural Network Deployment for Low-Power Embedded Systems

The methodology efficiently deploys and accelerates floating-point neural networks on embedded systems, optimizing performance, energy consumption, and hardware utilization.

Abstraction levels:

1. Model deployment
2. System infrastructure
3. **Streamlined acceleration**
4. Optimized processing

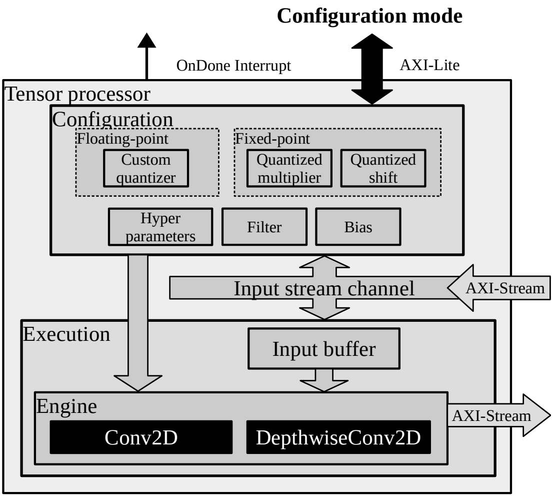


Trans-Precision Neural Network Deployment for Low-Power Embedded Systems

The methodology efficiently deploys and accelerates floating-point neural networks on embedded systems, optimizing performance, energy consumption, and hardware utilization.

Abstraction levels:

1. Model deployment
2. System infrastructure
3. **Streamlined acceleration**
4. Optimized processing

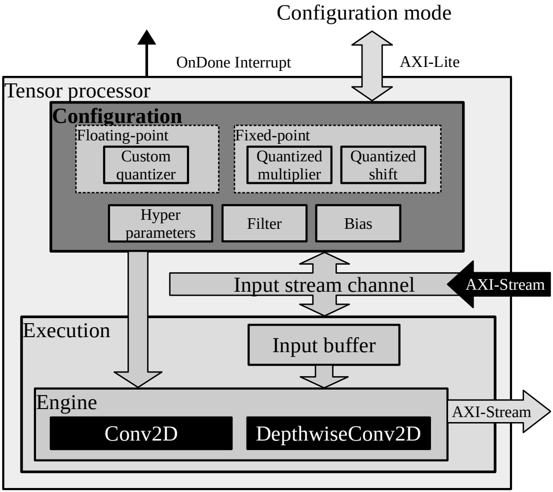


Trans-Precision Neural Network Deployment for Low-Power Embedded Systems

The methodology efficiently deploys and accelerates floating-point neural networks on embedded systems, optimizing performance, energy consumption, and hardware utilization.

Abstraction levels:

1. Model deployment
2. System infrastructure
3. **Streamlined acceleration**
4. Optimized processing

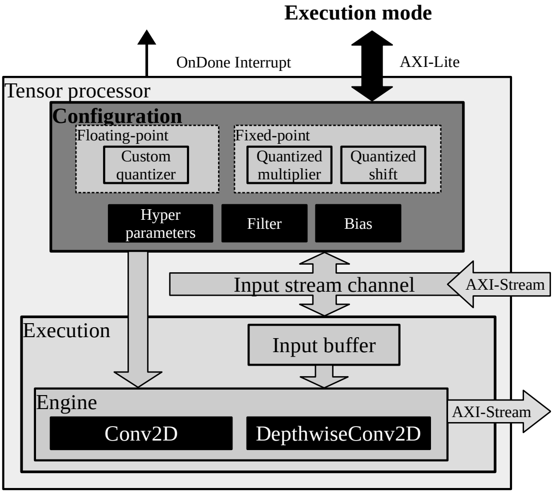


Trans-Precision Neural Network Deployment for Low-Power Embedded Systems

The methodology efficiently deploys and accelerates floating-point neural networks on embedded systems, optimizing performance, energy consumption, and hardware utilization.

Abstraction levels:

1. Model deployment
2. System infrastructure
3. **Streamlined acceleration**
4. Optimized processing

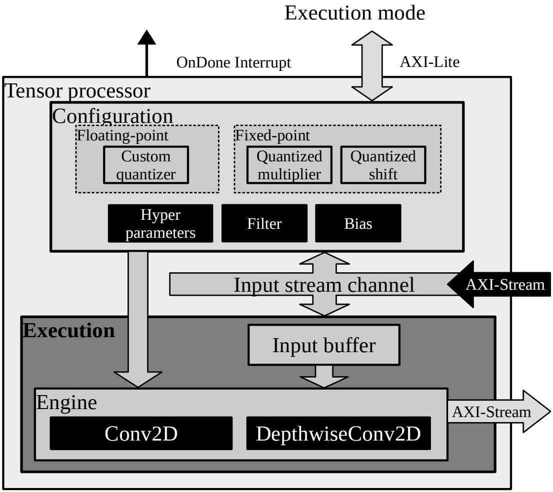


Trans-Precision Neural Network Deployment for Low-Power Embedded Systems

The methodology efficiently deploys and accelerates floating-point neural networks on embedded systems, optimizing performance, energy consumption, and hardware utilization.

Abstraction levels:

1. Model deployment
2. System infrastructure
3. **Streamlined acceleration**
4. Optimized processing

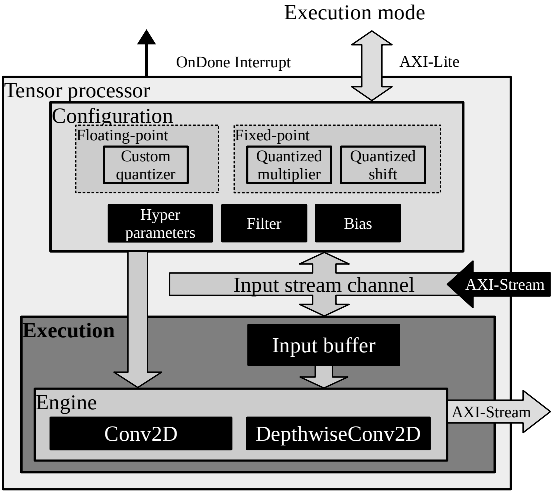


Trans-Precision Neural Network Deployment for Low-Power Embedded Systems

The methodology efficiently deploys and accelerates floating-point neural networks on embedded systems, optimizing performance, energy consumption, and hardware utilization.

Abstraction levels:

1. Model deployment
2. System infrastructure
3. **Streamlined acceleration**
4. Optimized processing

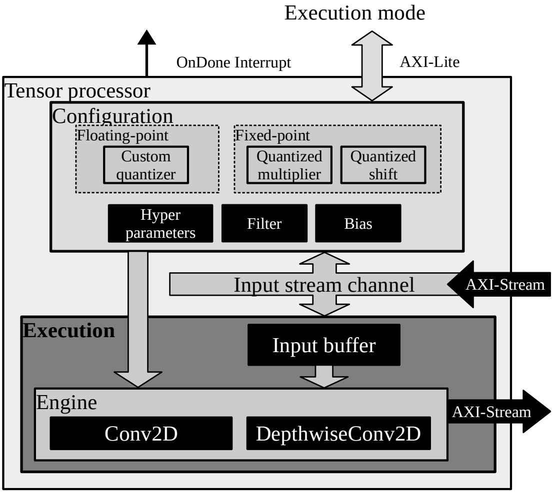


Trans-Precision Neural Network Deployment for Low-Power Embedded Systems

The methodology efficiently deploys and accelerates floating-point neural networks on embedded systems, optimizing performance, energy consumption, and hardware utilization.

Abstraction levels:

1. Model deployment
2. System infrastructure
3. **Streamlined acceleration**
4. Optimized processing

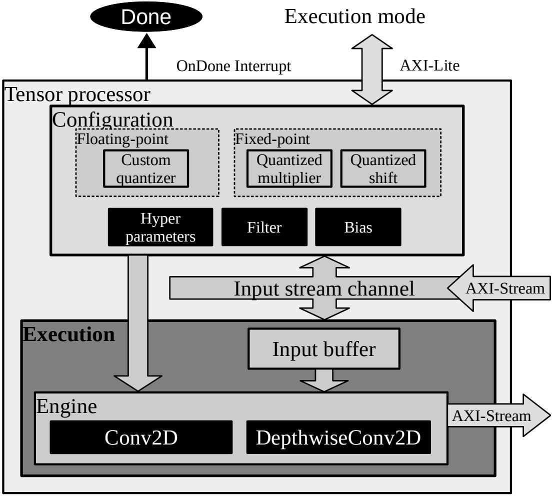


Trans-Precision Neural Network Deployment for Low-Power Embedded Systems

The methodology efficiently deploys and accelerates floating-point neural networks on embedded systems, optimizing performance, energy consumption, and hardware utilization.

Abstraction levels:

1. Model deployment
2. System infrastructure
3. **Streamlined acceleration**
4. Optimized processing

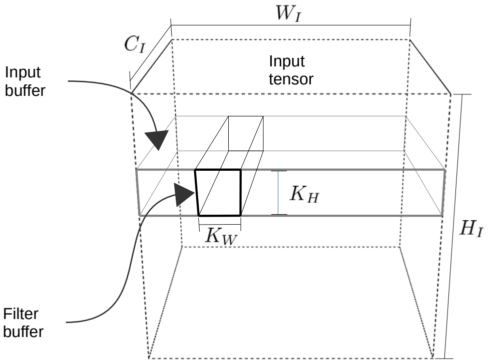


Trans-Precision Neural Network Deployment for Low-Power Embedded Systems

The methodology efficiently deploys and accelerates floating-point neural networks on embedded systems, optimizing performance, energy consumption, and hardware utilization.

Abstraction levels:

1. Model deployment
2. System infrastructure
3. Streamlined acceleration
4. **Optimized processing**



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

Trans-Precision Neural Network Deployment for Low-Power Embedded Systems

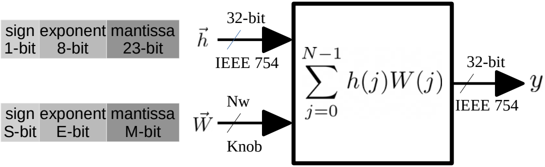
The methodology efficiently deploys and accelerates floating-point neural networks on embedded systems, optimizing performance, energy consumption, and hardware utilization.

Abstraction levels:

1. Model deployment
2. System infrastructure
3. Streamlined acceleration

4. Optimized processing

Multiply-Accumulate Unit
Hybrid Custom floating-point computation



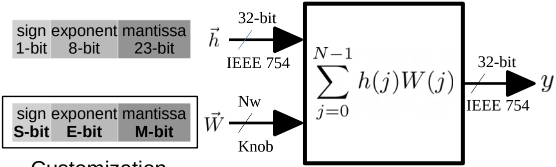
Trans-Precision Neural Network Deployment for Low-Power Embedded Systems

The methodology efficiently deploys and accelerates floating-point neural networks on embedded systems, optimizing performance, energy consumption, and hardware utilization.

Abstraction levels:

1. Model deployment
2. System infrastructure
3. Streamlined acceleration
4. **Optimized processing**

Multiply-Accumulate Unit
Hybrid Custom floating-point computation



Customization

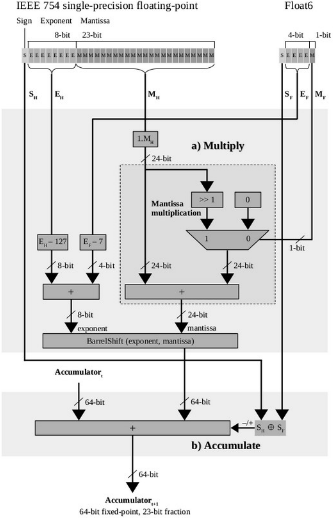
Methodology

Trans-Precision Neural Network Deployment for Low-Power Embedded Systems

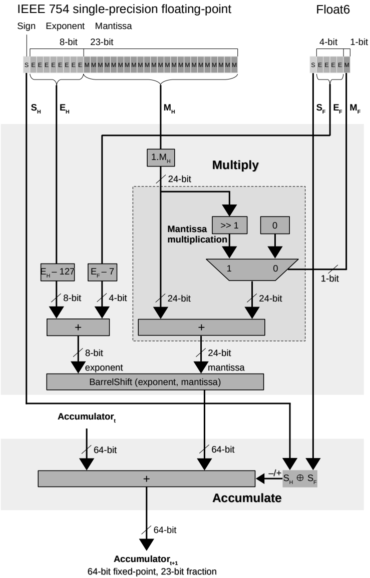
The methodology efficiently deploys and accelerates floating-point neural networks on embedded systems, optimizing performance, energy consumption, and hardware utilization.

Abstraction levels:

1. Model deployment
2. System infrastructure
3. Streamlined acceleration
4. **Optimized processing**



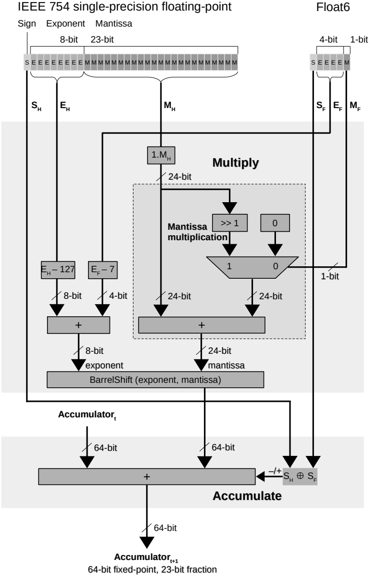
Hybrid Custom Floating-Point Multiply-Accumulate Unit



Hybrid Custom Floating-Point Multiply-Accumulate Unit

Vector dot-product:

- Multiplication
- Accumulation



Hybrid Custom Floating-Point Multiply-Accumulate Unit

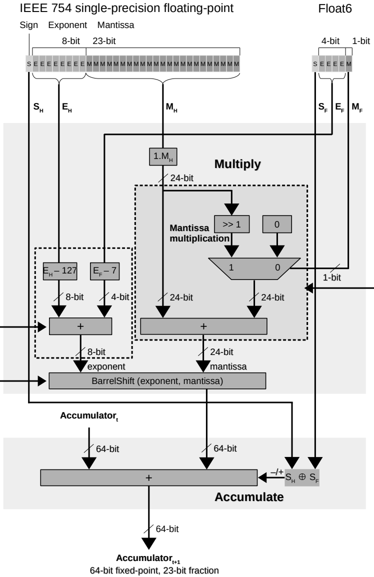
Vector dot-product:

Multiplication

Accumulation

Exponent addition

Denormalization



1-bit mantissa multiplication

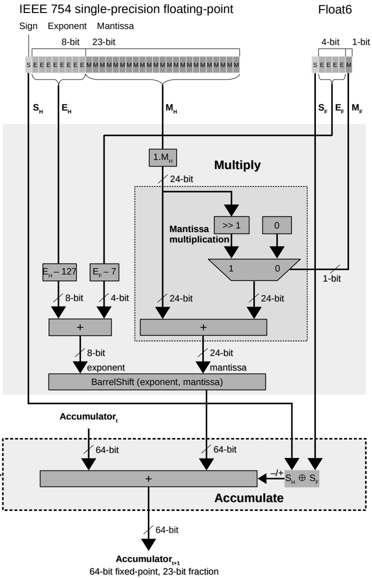
Hybrid Custom Floating-Point Multiply-Accumulate Unit

Vector dot-product:

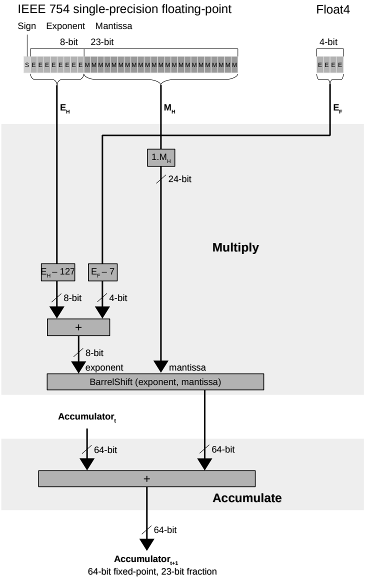
Multiplication

Accumulation

Denormalized accumulation →



Hybrid Custom Floating-Point Multiply-Accumulate Unit



Hybrid Custom Floating-Point Multiply-Accumulate Unit

For **noise-robust applications** with **non-negativity constraints**.

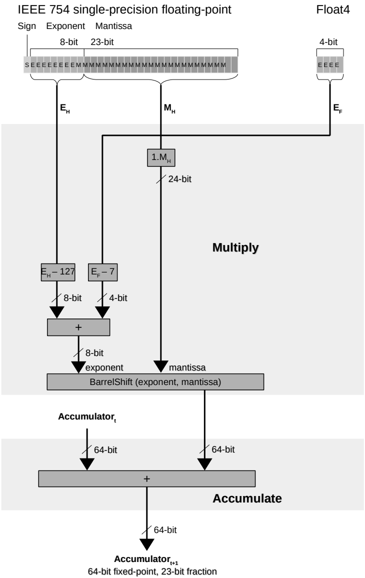
Non-negative Tensor Factorization

Non-negative Sparse Coding

Non-negative Matrix Factorization

Spike-by-Spike Neural Networks

Etc.



Custom Floating-Point Quantization

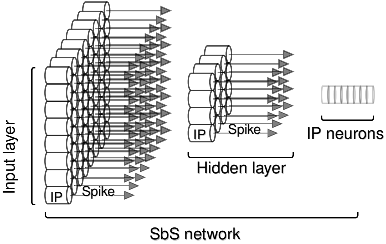
Algorithm 1: Custom floating-point quantization

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`
              $exp \leftarrow exp + 1$
          $x \leftarrow (-1)^{sign} \cdot 2^{exp} \cdot (1 + cman \cdot 2^{-M_{size}})$
    $SetWeights(layer, filter, bias)$

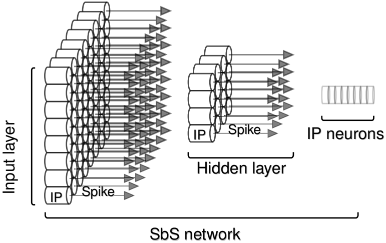
Spike-by-Spike Neural Network

Structure

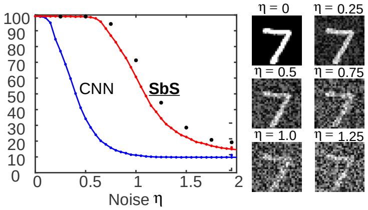


Spike-by-Spike Neural Network

Structure

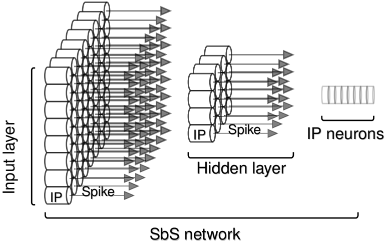


Noise robustness

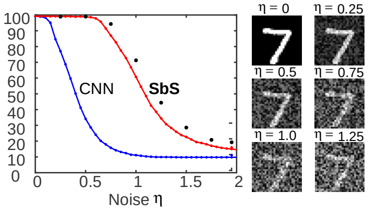


Spike-by-Spike Neural Network

Structure



Noise robustness

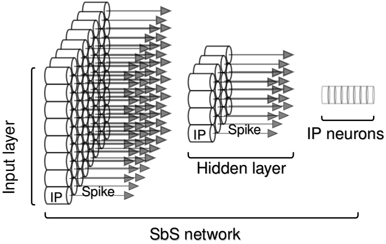


Neuron update

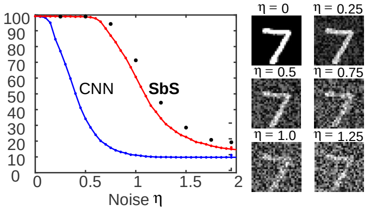
$$h_\mu^{new}(i) = \frac{1}{1+\epsilon}\left(h_\mu(i) + \epsilon \frac{h_\mu(i) W(s_t|i)}{\sum_j h_\mu(j) W(s_t|j)}\right)$$

Spike-by-Spike Neural Network

Structure



Noise robustness



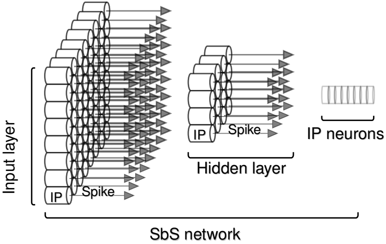
Neuron update

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

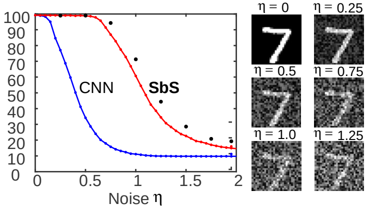
Non-negativity $\quad 0 \leq W(s_t|j) \leq 1$

Spike-by-Spike Neural Network

Structure



Noise robustness



Neuron update

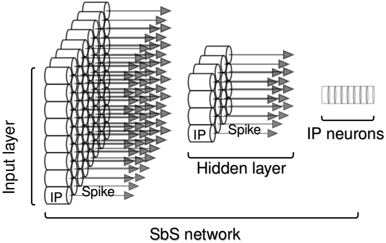
$$h_\mu^{new}(i) = \frac{1}{1+\epsilon}\left(h_\mu(i) + \epsilon \frac{h_\mu(i)W(s_t|i)}{\sum_j h_\mu(j)W(s_t|j)}\right)$$

Non-negativity $0 \leq W(s_t|j) \leq 1$

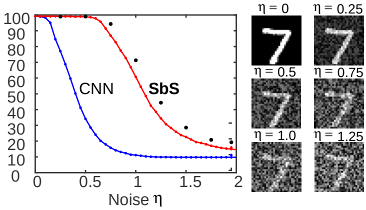
Normalized $\sum_{s_t=0}^{M-1} W(s_t|j) = 1$

Spike-by-Spike Neural Network

Structure



Noise robustness



Properties:

- Noise robustness
- Iterative optimization
- No sign bit required
- Requires division
- Compute and memory intensive

Neuron update

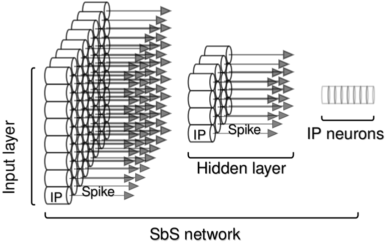
$$h_\mu^{new}(i) = \frac{1}{1+\epsilon}\left(h_\mu(i) + \epsilon \frac{h_\mu(i)W(s_t|i)}{\sum_j h_\mu(j)W(s_t|j)}\right)$$

Non-negativity $0 \leq W(s_t|j) \leq 1$

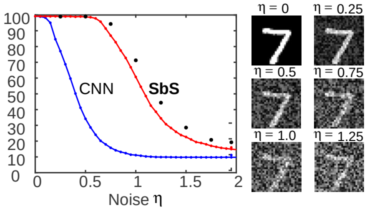
Normalized $\sum_{s_t=0}^{M-1} W(s_t|j) = 1$

Spike-by-Spike Neural Network

Structure



Noise robustness



Properties:

- **Noise robustness**
- Iterative optimization
- No sign bit required
- Requires division
- Compute and memory intensive

Neuron update

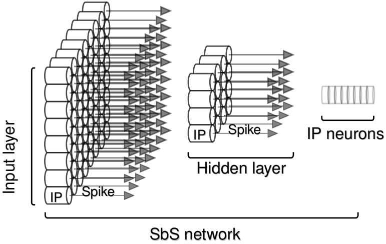
$$h_\mu^{new}(i) = \frac{1}{1+\epsilon}\left(h_\mu(i) + \epsilon \frac{h_\mu(i) W(s_t|i)}{\sum_j h_\mu(j) W(s_t|j)}\right)$$

Non-negativity $0 \leq W(s_t|j) \leq 1$

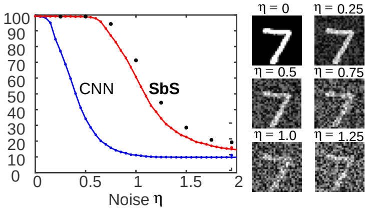
Normalized $\sum_{s_t=0}^{M-1} W(s_t|j) = 1$

Spike-by-Spike Neural Network

Structure



Noise robustness



Properties:

- Noise robustness
- **Iterative optimization**
- No sign bit required
- Requires division
- Compute and memory intensive

Neuron update

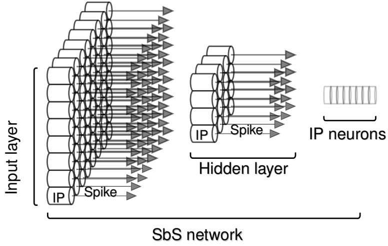
$$h_\mu^{new}(i) = \frac{1}{1+\epsilon}\left(h_\mu(i) + \epsilon \frac{h_\mu(i)W(s_t|i)}{\sum_j h_\mu(j)W(s_t|j)}\right)$$

Non-negativity $0 \leq W(s_t|j) \leq 1$

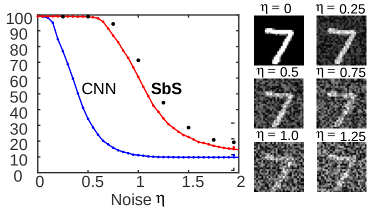
Normalized $\sum_{s_t=0}^{M-1} W(s_t|j) = 1$

Spike-by-Spike Neural Network

Structure



Noise robustness



Properties:

- Noise robustness
- Iterative optimization
- **No sign bit required**
- Requires division
- Compute and memory intensive

Neuron update

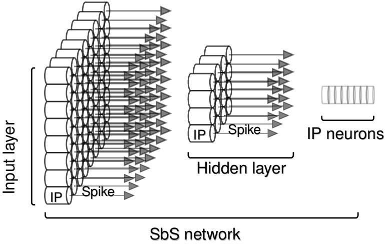
$$h_\mu^{new}(i) = \frac{1}{1+\epsilon}\left(h_\mu(i) + \epsilon \frac{h_\mu(i)W(s_t|i)}{\sum_j h_\mu(j)W(s_t|j)}\right)$$

Non-negativity $0 \leq W(s_t|j) \leq 1$

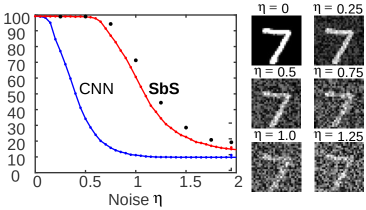
Normalized $\sum_{s_t=0}^{M-1} W(s_t|j) = 1$

Spike-by-Spike Neural Network

Structure



Noise robustness



Properties:

- Noise robustness
- Iterative optimization
- No sign bit required
- **Requires division**
- Compute and memory intensive

Neuron update

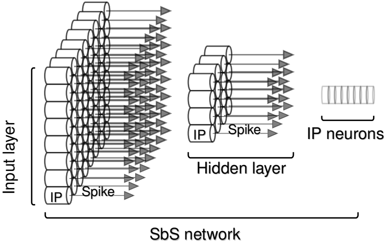
$$h_\mu^{new}(i) = \frac{1}{1+\epsilon}\left(h_\mu(i) + \epsilon \frac{h_\mu(i)W(s_t|i)}{\sum_j h_\mu(j)W(s_t|j)}\right)$$

Non-negativity $0 \leq W(s_t|j) \leq 1$

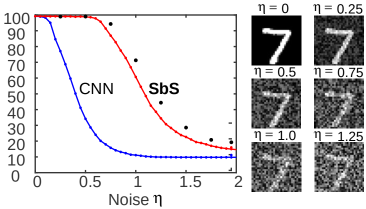
Normalized $\sum_{s_t=0}^{M-1} W(s_t|j) = 1$

Spike-by-Spike Neural Network

Structure



Noise robustness



Properties:

- Noise robustness
- Iterative optimization
- No sign bit required
- Requires division
- **Compute and memory intensive**

Neuron update

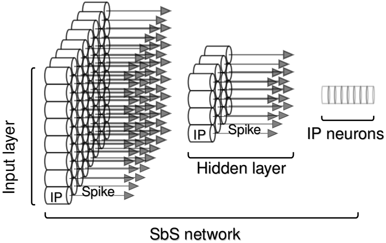
$$h_\mu^{new}(i) = \frac{1}{1+\epsilon}\left(h_\mu(i) + \epsilon \frac{h_\mu(i)W(s_t|i)}{\sum_j h_\mu(j)W(s_t|j)}\right)$$

Non-negativity $0 \leq W(s_t|j) \leq 1$

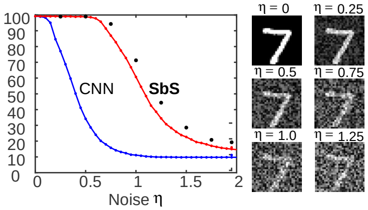
Normalized $\sum_{s_t=0}^{M-1} W(s_t|j) = 1$

Spike-by-Spike Neural Network

Structure



Noise robustness



Properties:

- Noise robustness
- Iterative optimization
- No sign bit required
- Requires division
- Compute and memory intensive

Neuron update

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

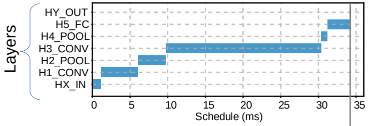
Non-negativity $\quad 0 \leq W(s_t|j) \leq 1$

Normalized $\quad \sum_{s_t=0}^{M-1} W(s_t|j) = 1$

Deployment

Floating-point 32-bit

ARM Cortex A9 @ 666 MHz

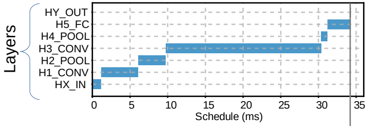


Iteration = 34.3 ms

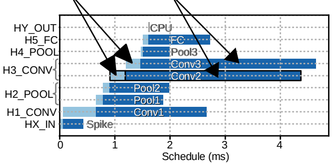
Deployment

Floating-point 32-bit

ARM Cortex A9 @ 666 MHz

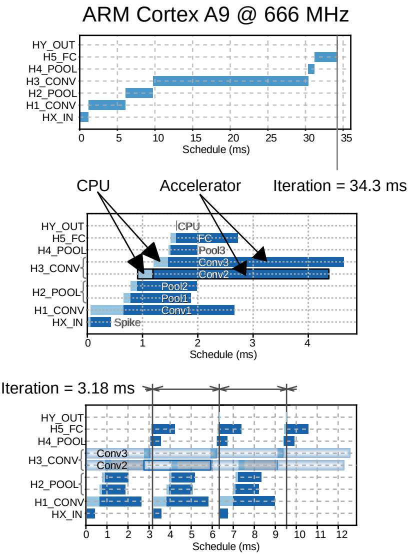


CPU Accelerator Iteration = 34.3 ms



Deployment

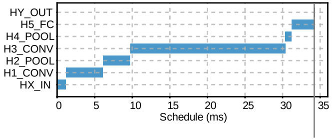
Floating-point 32-bit



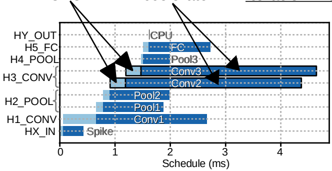
Deployment

Floating-point 32-bit

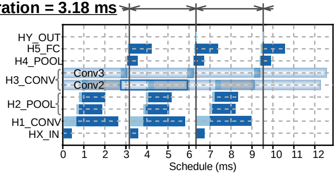
ARM Cortex A9 @ 666 MHz



CPU Accelerator **Iteration = 34.3 ms**

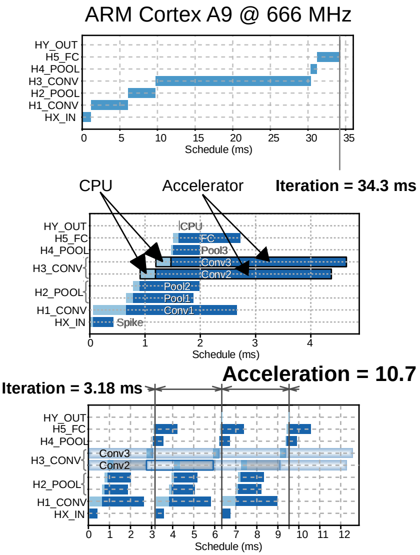


Iteration = 3.18 ms



Deployment

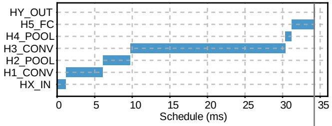
Floating-point 32-bit



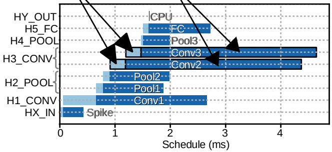
Deployment

Floating-point 32-bit

ARM Cortex A9 @ 666 MHz

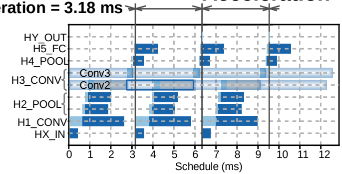


CPU Accelerator **Iteration = 34.3 ms**

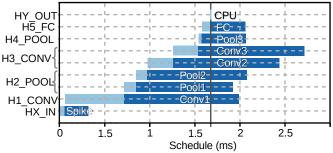


Acceleration = 10.7

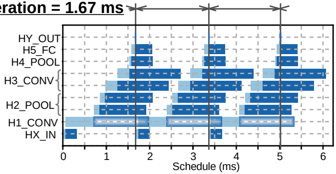
Iteration = 3.18 ms



Hybrid logarithmic 4-bit



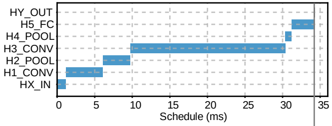
Iteration = 1.67 ms



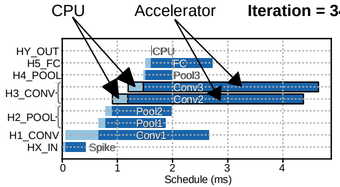
Deployment

Floating-point 32-bit

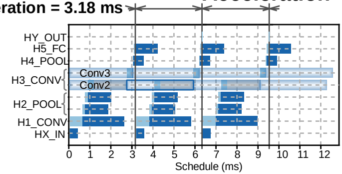
ARM Cortex A9 @ 666 MHz



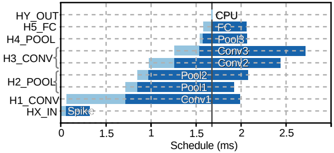
Iteration = 34.3 ms



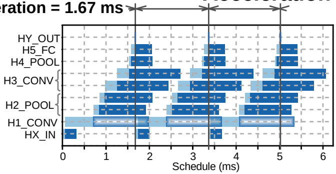
Iteration = 3.18 ms **Acceleration = 10.7**



Hybrid logarithmic 4-bit

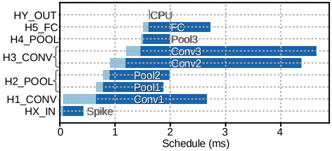


Iteration = 1.67 ms **Acceleration = 20.5**

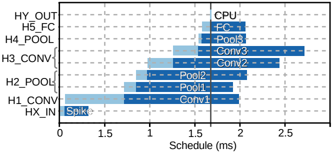


Deployment

Standard floating-point

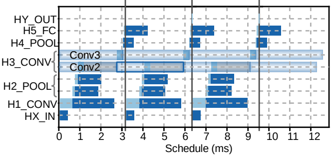


Hybrid logarithmic 4-bit



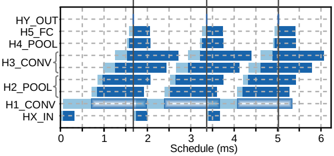
Acceleration = 10.7

Iteration = 3.18 ms

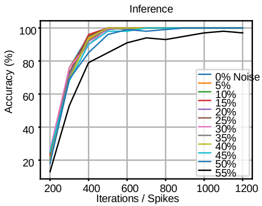


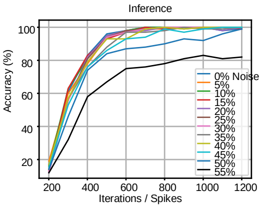
Acceleration = 20.5

Iteration = 1.67 ms

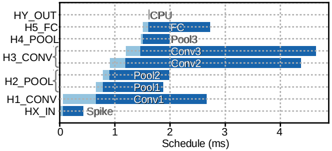


Deployment

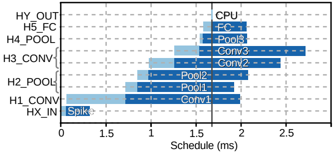




Standard floating-point

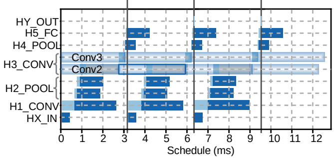


Hybrid logarithmic 4-bit



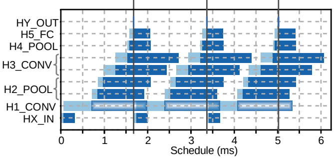
Acceleration = 10.7

Iteration = 3.18 ms



Acceleration = 20.5

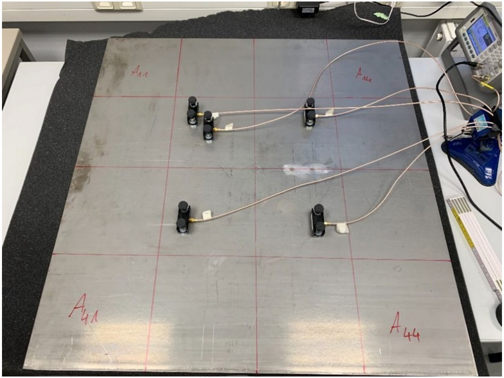
Iteration = 1.67 ms



TinyML Application: CNN Sensor Analytics for Structural Health Monitoring

Experimental setup:

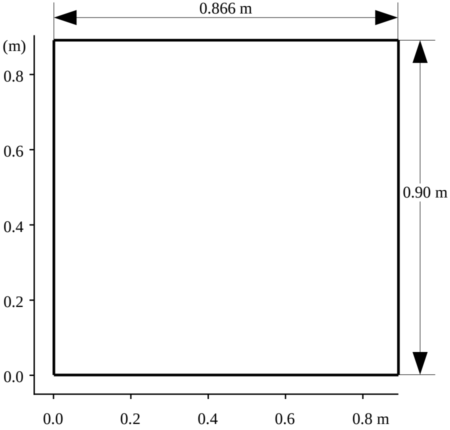
- Sensor and noise positions
- Training dataset
- Testing dataset
- Model architecture



TinyML Application: CNN Sensor Analytics for Structural Health Monitoring

Experimental setup:

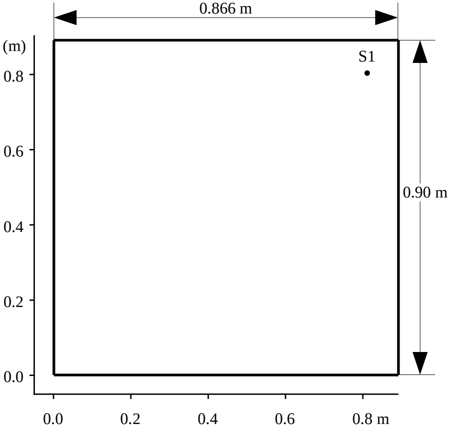
- **Sensor and noise positions**
- Training dataset
- Testing dataset
- Model architecture



TinyML Application: CNN Sensor Analytics for Structural Health Monitoring

Experimental setup:

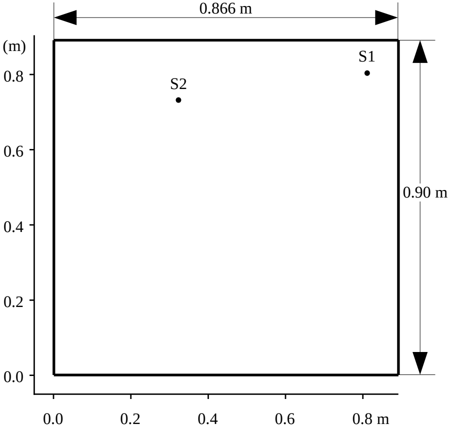
- **Sensor and noise positions**
- Training dataset
- Testing dataset
- Model architecture



TinyML Application: CNN Sensor Analytics for Structural Health Monitoring

Experimental setup:

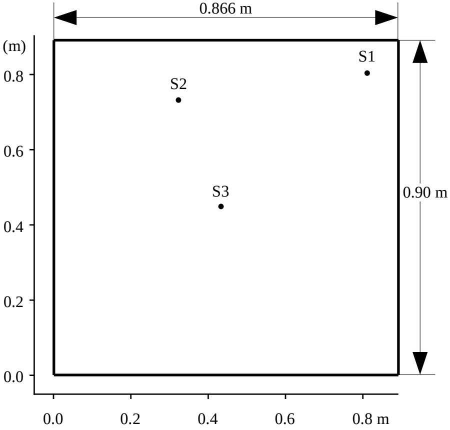
- **Sensor and noise positions**
- Training dataset
- Testing dataset
- Model architecture



TinyML Application: CNN Sensor Analytics for Structural Health Monitoring

Experimental setup:

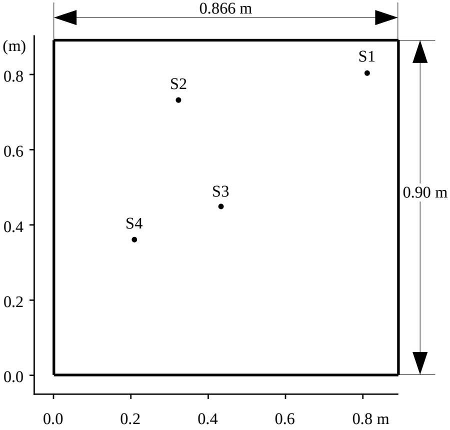
- **Sensor and noise positions**
- Training dataset
- Testing dataset
- Model architecture



TinyML Application: CNN Sensor Analytics for Structural Health Monitoring

Experimental setup:

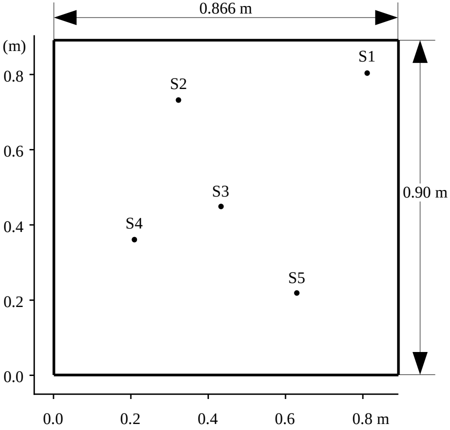
- **Sensor and noise positions**
- Training dataset
- Testing dataset
- Model architecture



TinyML Application: CNN Sensor Analytics for Structural Health Monitoring

Experimental setup:

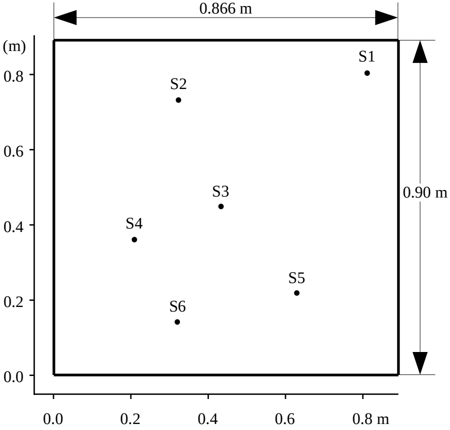
- **Sensor and noise positions**
- Training dataset
- Testing dataset
- Model architecture



TinyML Application: CNN Sensor Analytics for Structural Health Monitoring

Experimental setup:

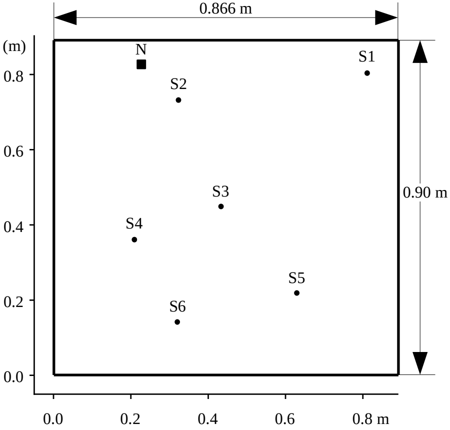
- **Sensor and noise positions**
- Training dataset
- Testing dataset
- Model architecture



TinyML Application: CNN Sensor Analytics for Structural Health Monitoring

Experimental setup:

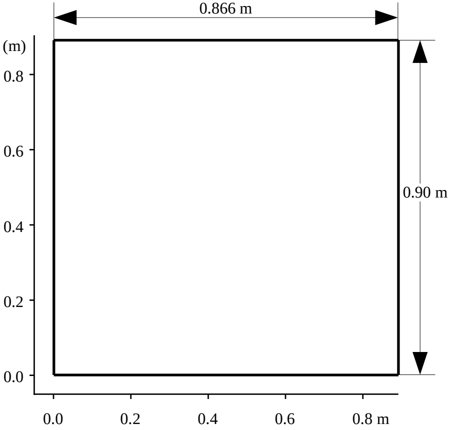
- **Sensor and noise positions**
- Training dataset
- Testing dataset
- Model architecture



TinyML Application: CNN Sensor Analytics for Structural Health Monitoring

Experimental setup:

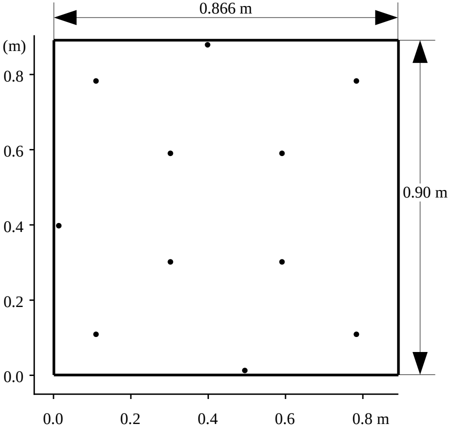
- Sensor and noise positions
- **Training dataset**
- Testing dataset
- Model architecture



TinyML Application: CNN Sensor Analytics for Structural Health Monitoring

Experimental setup:

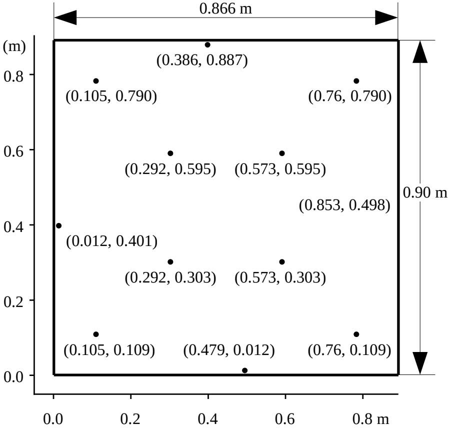
- Sensor and noise positions
- **Training dataset**
- Testing dataset
- Model architecture



TinyML Application: CNN Sensor Analytics for Structural Health Monitoring

Experimental setup:

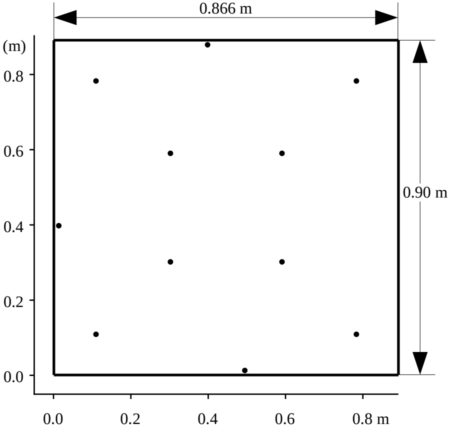
- Sensor and noise positions
- **Training dataset**
- Testing dataset
- Model architecture



TinyML Application: CNN Sensor Analytics for Structural Health Monitoring

Experimental setup:

- Sensor and noise positions
- **Training dataset**
- Testing dataset
- Model architecture



TinyML Application: CNN Sensor Analytics for Structural Health Monitoring

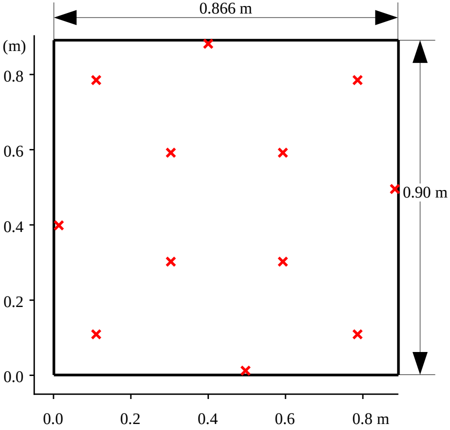
Experimental setup:

Sensor and noise positions

Training dataset

Testing dataset

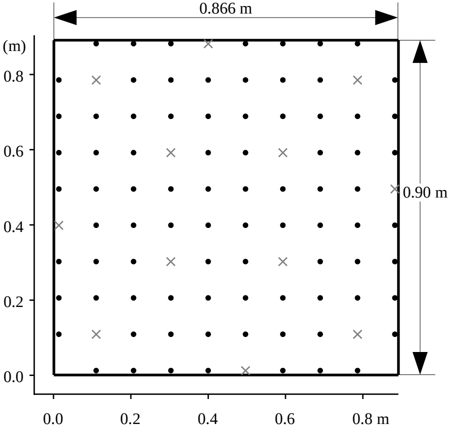
Model architecture



TinyML Application: CNN Sensor Analytics for Structural Health Monitoring

Experimental setup:

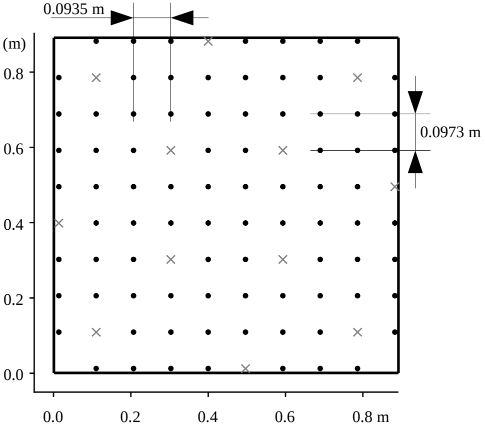
- Sensor and noise positions
- Training dataset
- **Testing dataset**
- Model architecture



TinyML Application: CNN Sensor Analytics for Structural Health Monitoring

Experimental setup:

- Sensor and noise positions
- Training dataset
- **Testing dataset**
- Model architecture

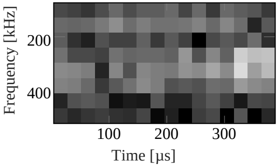


TinyML Application: CNN Sensor Analytics for Structural Health Monitoring

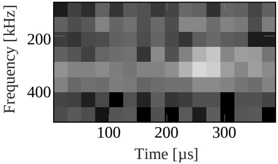
Experimental setup:

- Sensor and noise positions
- Training dataset
- Testing dataset
- **Model architecture**

S1



S2

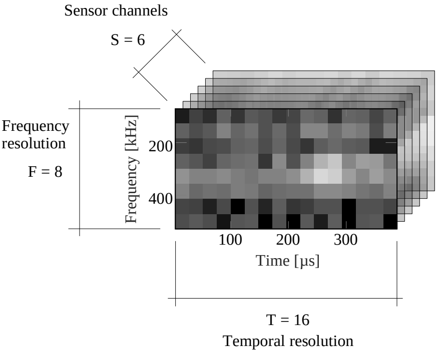


TinyML Application: CNN Sensor Analytics for Structural Health Monitoring

Experimental setup:

- Sensor and noise positions
- Training dataset
- Testing dataset
- **Model architecture**

Input tensor



TinyML Application: CNN Sensor Analytics for Structural Health Monitoring

Experimental setup:

- Sensor and noise positions
- Training dataset
- Testing dataset
- **Model architecture**

CNN-regression model

Output tensor (x, y)

FC (2), Linear
FC (E = 64), ReLu
FC (D = 196), ReLu
Flatten
2 x 2 MaxPool, stride 2
BatchNormalization
3 x 3 Conv (C = 60), ReLu
2 x 2 MaxPool, stride 2
BatchNormalization
3 x 3 Conv (B = 55), ReLu
2 x 2 MaxPool, stride 2
BatchNormalization
3 x 3 Conv (A = 50), ReLu
Input tensor (F x T x S)

Boosting TinyML Performance with Hybrid Floating-Point 6-bit

TinyML Application: CNN Sensor Analytics for Structural Health Monitoring

Experimental setup:

- Sensor and noise positions
- Training dataset
- Testing dataset
- **Model architecture**

CNN-regression model

Output tensor (x, y)

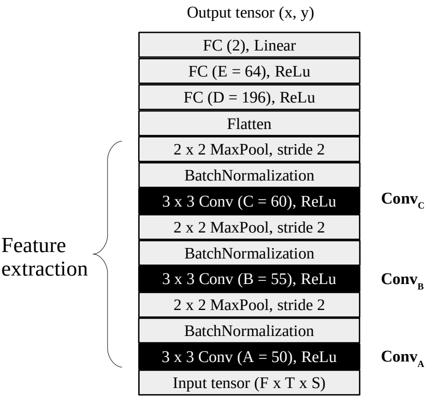
Layer	
FC (2), Linear	
FC (E = 64), ReLu	
FC (D = 196), ReLu	
Flatten	
2 x 2 MaxPool, stride 2	
BatchNormalization	
3 x 3 Conv (C = 60), ReLu	**Conv_C**
2 x 2 MaxPool, stride 2	
BatchNormalization	
3 x 3 Conv (B = 55), ReLu	**Conv_B**
2 x 2 MaxPool, stride 2	
BatchNormalization	
3 x 3 Conv (A = 50), ReLu	**Conv_A**
Input tensor (F x T x S)	

TinyML Application: CNN Sensor Analytics for Structural Health Monitoring

Experimental setup:

- Sensor and noise positions
- Training dataset
- Testing dataset
- **Model architecture**

CNN-regression model

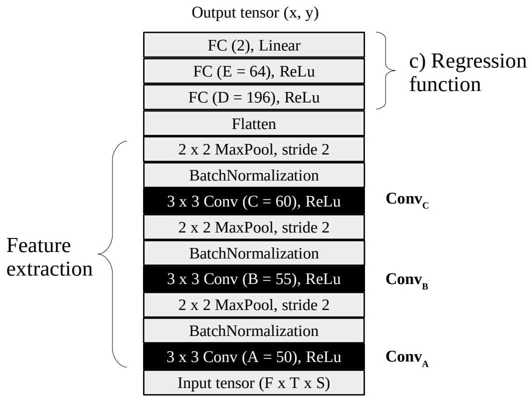


TinyML Application: CNN Sensor Analytics for Structural Health Monitoring

Experimental setup:

- Sensor and noise positions
- Training dataset
- Testing dataset
- **Model architecture**

CNN-regression model



Boosting TinyML Performance with Hybrid Floating-Point 6-bit

TinyML Application: CNN Sensor Analytics for Structural Health Monitoring

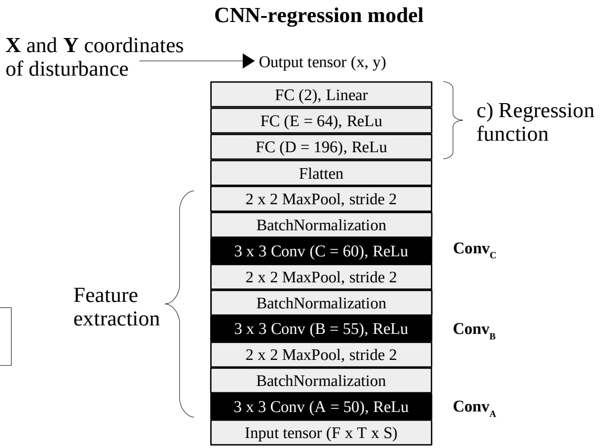
Experimental setup:

Sensor and noise positions

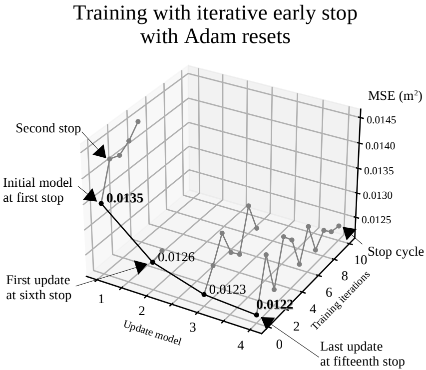
Training dataset

Testing dataset

Model architecture

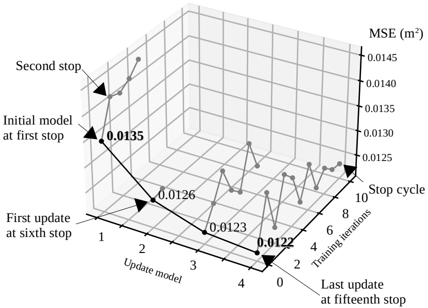


Multi-Phase Model Optimization

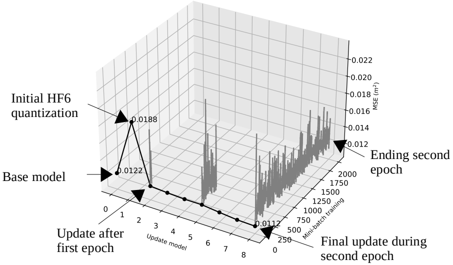


Multi-Phase Model Optimization

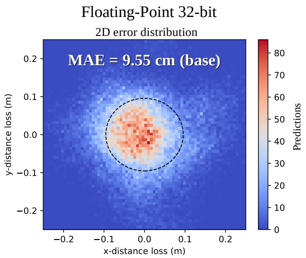
Training with iterative early stop with Adam resets



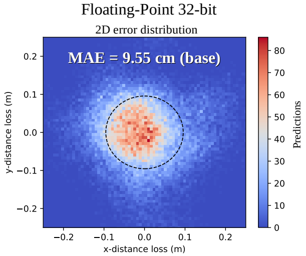
Quantization-aware training for hybrid floating-point 6-bit

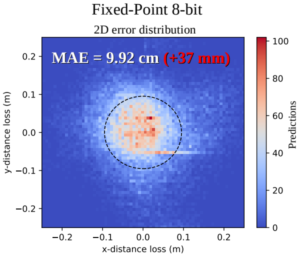


Assessing X and Y Coordinate Prediction Accuracy Across Multiple Quantization Strategies

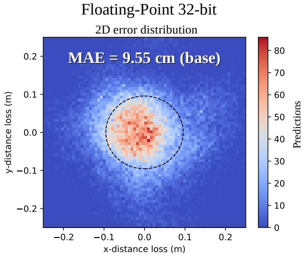


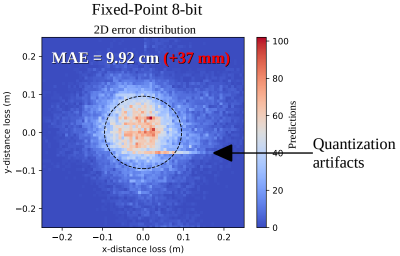
Assessing X and Y Coordinate Prediction Accuracy Across Multiple Quantization Strategies





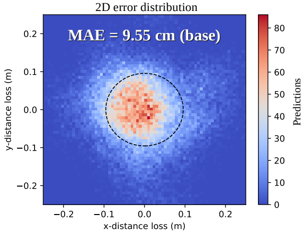
Assessing X and Y Coordinate Prediction Accuracy Across Multiple Quantization Strategies



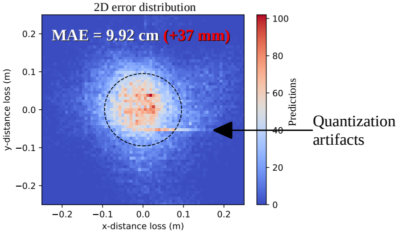


Assessing X and Y Coordinate Prediction Accuracy Across Multiple Quantization Strategies

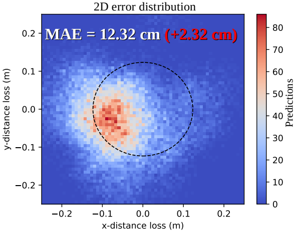
Floating-Point 32-bit



Fixed-Point 8-bit

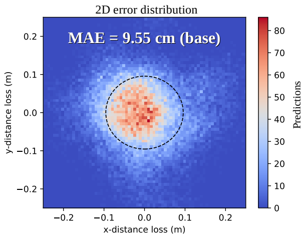


Hybrid-Float6 (E4M1) without QAT

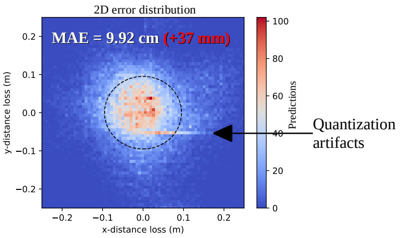


Assessing X and Y Coordinate Prediction Accuracy Across Multiple Quantization Strategies

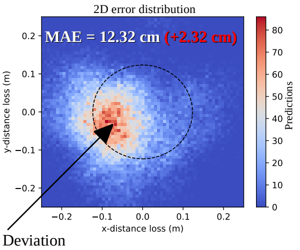
Floating-Point 32-bit



Fixed-Point 8-bit

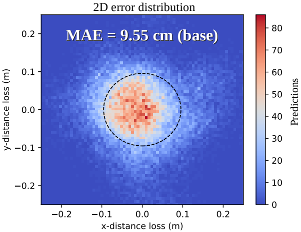


Hybrid-Float6 (E4M1) without QAT

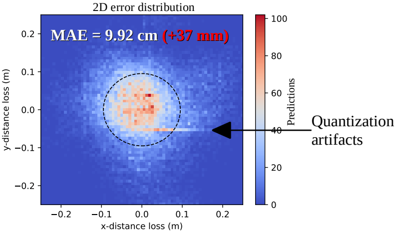


Assessing X and Y Coordinate Prediction Accuracy Across Multiple Quantization Strategies

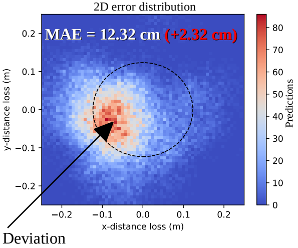
Floating-Point 32-bit



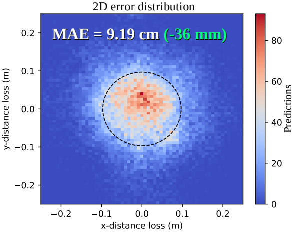
Fixed-Point 8-bit



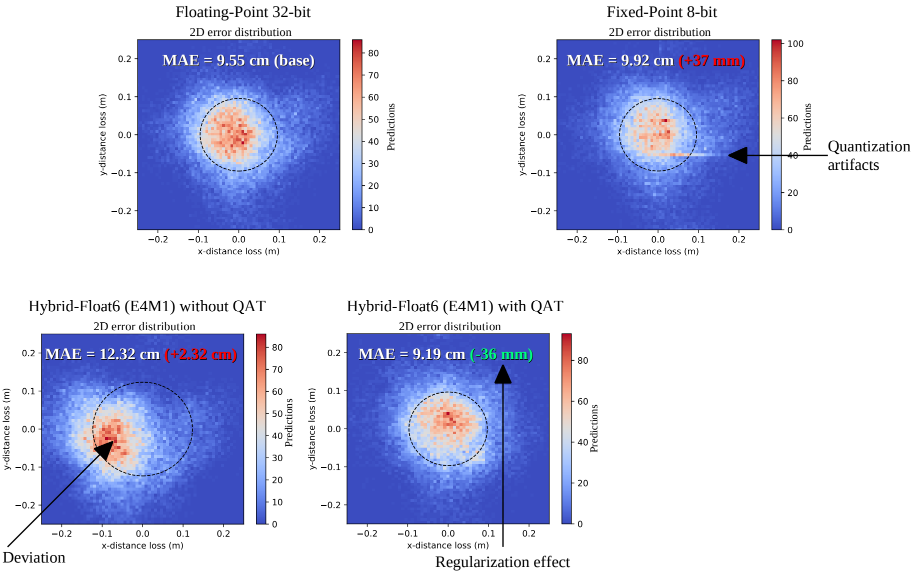
Hybrid-Float6 (E4M1) without QAT



Hybrid-Float6 (E4M1) with QAT

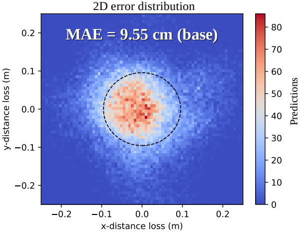


Assessing X and Y Coordinate Prediction Accuracy Across Multiple Quantization Strategies

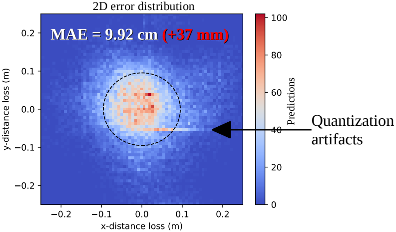


Assessing X and Y Coordinate Prediction Accuracy Across Multiple Quantization Strategies

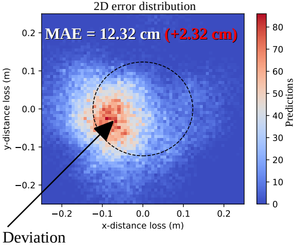
Floating-Point 32-bit



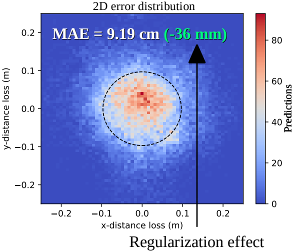
Fixed-Point 8-bit



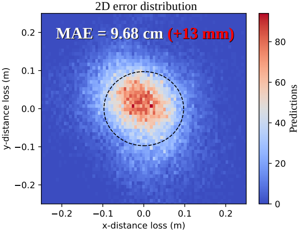
Hybrid-Float6 (E4M1) without QAT



Hybrid-Float6 (E4M1) with QAT

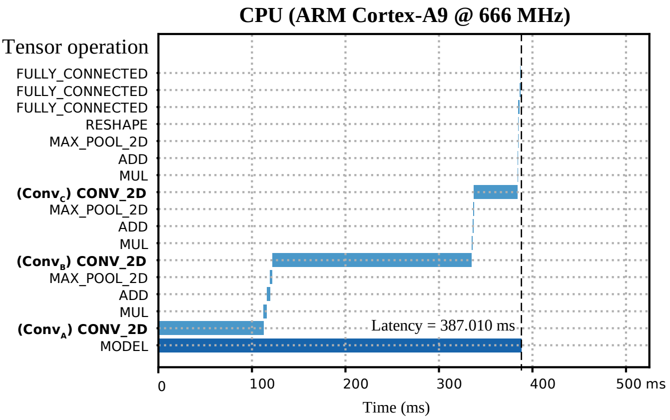


Hybrid-Logarithmic 6-bit (E5M0) with QAT



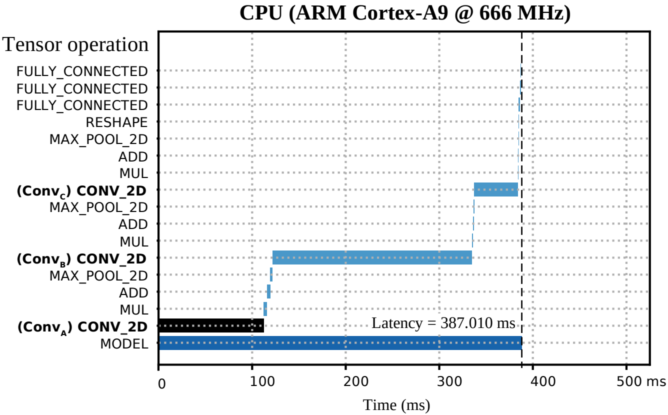
Runtime Profiling: Latencies and Scheduling on Floating-Point and Hybrid Float 6 Accelerators

Tensor Op	CPU (ms)	TP FP32 (ms)	Gain	**TP HF6 (ms)**	**Gain**
Conv_A					
Conv_B					
Conv_C					



Runtime Profiling: Latencies and Scheduling on Floating-Point and Hybrid Float 6 Accelerators

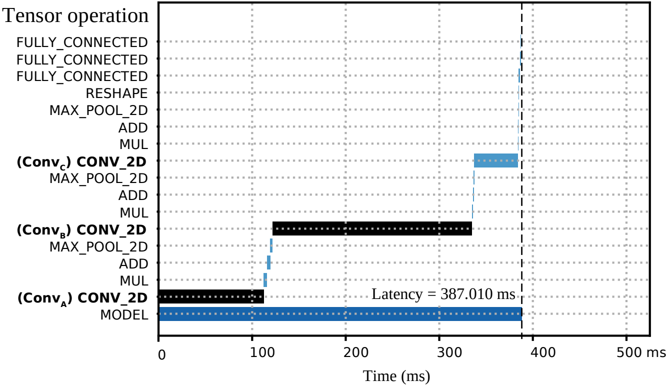
Tensor Op	CPU (ms)	TP FP32 (ms)	Gain	**TP HF6 (ms)**	**Gain**
Conv_A	112.24				
Conv_B					
Conv_C					



Runtime Profiling: Latencies and Scheduling on Floating-Point and Hybrid Float 6 Accelerators

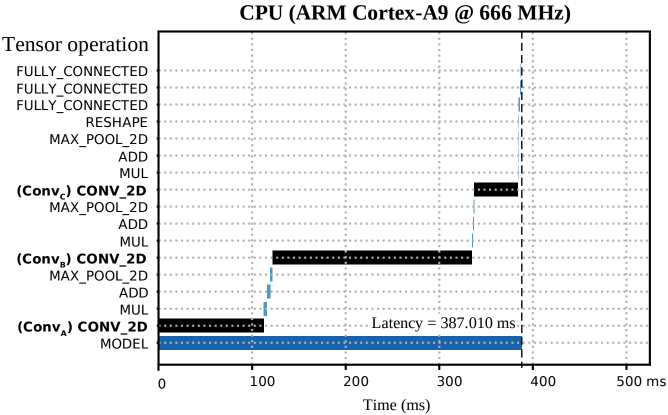
Tensor Op	CPU (ms)	TP FP32 (ms)	Gain	**TP HF6 (ms)**	**Gain**
Conv_A	112.24				
Conv_B	213.13				
Conv_C					

CPU (ARM Cortex-A9 @ 666 MHz)



Runtime Profiling: Latencies and Scheduling on Floating-Point and Hybrid Float 6 Accelerators

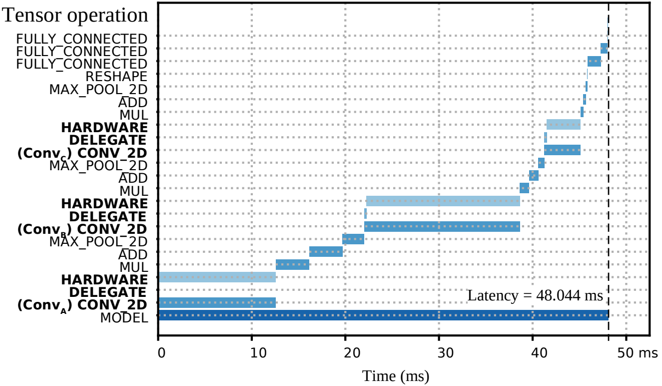
Tensor Op	CPU (ms)	TP FP32 (ms)	Gain	**TP HF6 (ms)**	**Gain**
Conv_A	112.24				
Conv_B	213.13				
Conv_C	46.59				



Runtime Profiling: Latencies and Scheduling on Floating-Point and Hybrid Float 6 Accelerators

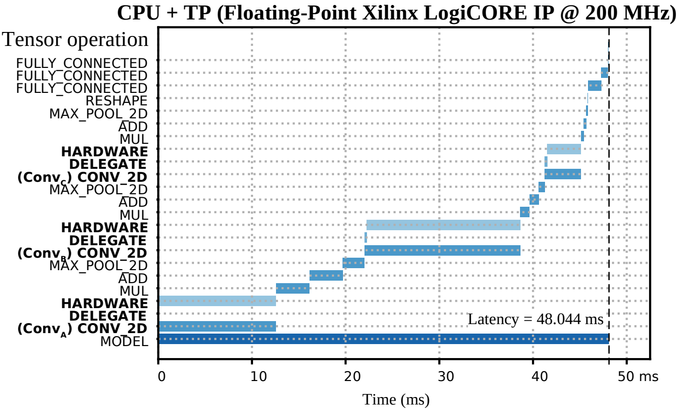
Tensor Op	CPU (ms)	TP FP32 (ms)	Gain	**TP HF6 (ms)**	**Gain**
Conv_A	112.24				
Conv_B	213.13				
Conv_C	46.59				

CPU + TP (Floating-Point Xilinx LogiCORE IP @ 200 MHz)



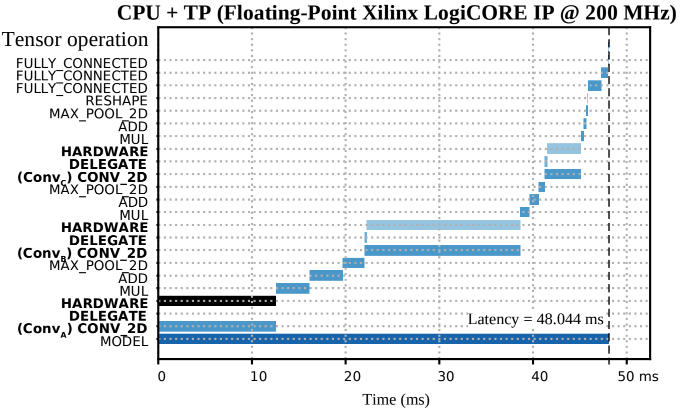
Runtime Profiling: Latencies and Scheduling on Floating-Point and Hybrid Float 6 Accelerators

Tensor Op	CPU (ms)	TP FP32 (ms)	Gain	**TP HF6 (ms)**	**Gain**
Conv_A	112.24				
Conv_B	213.13				
Conv_C	46.59				



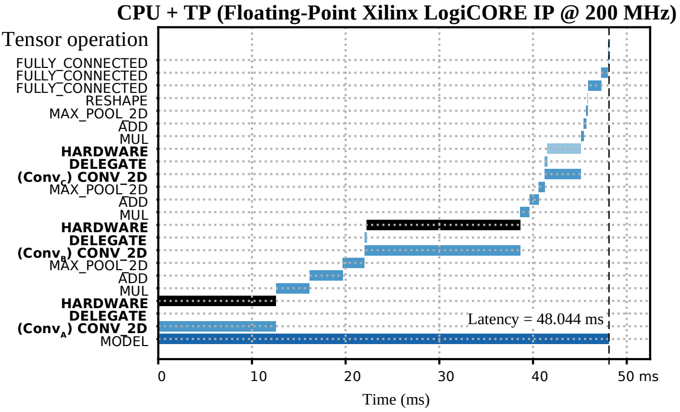
Runtime Profiling: Latencies and Scheduling on Floating-Point and Hybrid Float 6 Accelerators

Tensor Op	CPU (ms)	TP FP32 (ms)	Gain	**TP HF6 (ms)**	**Gain**
$Conv_A$	112.24	12.49	8.9		
$Conv_B$	213.13				
$Conv_C$	46.59				



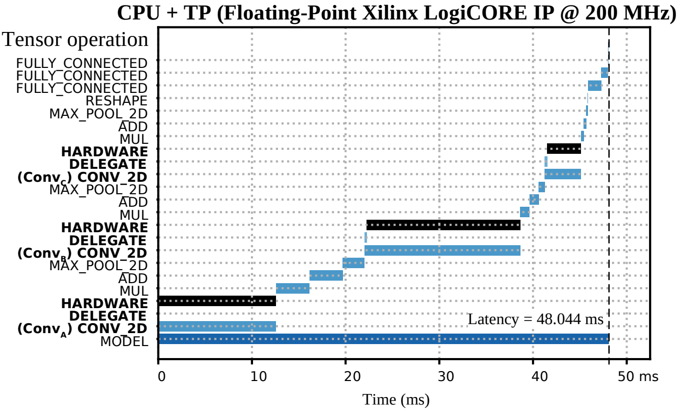
Runtime Profiling: Latencies and Scheduling on Floating-Point and Hybrid Float 6 Accelerators

Tensor Op	CPU (ms)	TP FP32 (ms)	Gain	**TP HF6 (ms)**	**Gain**
$Conv_A$	112.24	12.49	8.9		
$Conv_B$	213.13	16.39	13.0		
$Conv_C$	46.59				



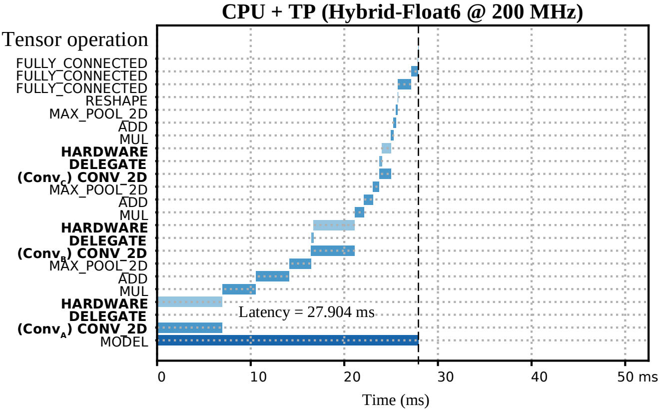
Runtime Profiling: Latencies and Scheduling on Floating-Point and Hybrid Float 6 Accelerators

Tensor Op	CPU (ms)	TP FP32 (ms)	Gain	**TP HF6 (ms)**	**Gain**
$Conv_A$	112.24	12.49	8.9		
$Conv_B$	213.13	16.39	13.0		
$Conv_C$	46.59	3.59	12.9		



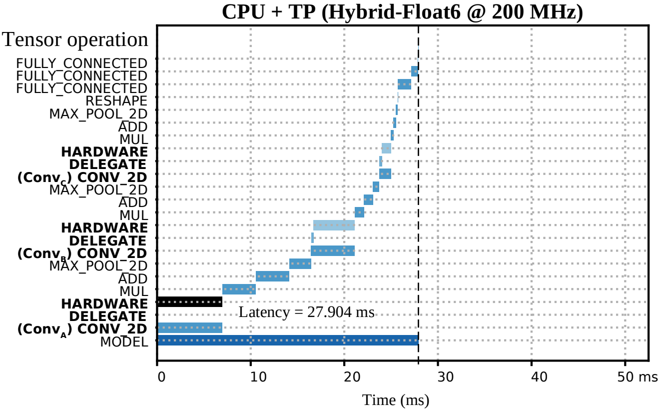
Runtime Profiling: Latencies and Scheduling on Floating-Point and Hybrid Float 6 Accelerators

Tensor Op	CPU (ms)	TP FP32 (ms)	Gain	**TP HF6 (ms)**	**Gain**
$Conv_A$	112.24	12.49	8.9		
$Conv_B$	213.13	16.39	13.0		
$Conv_C$	46.59	3.59	12.9		



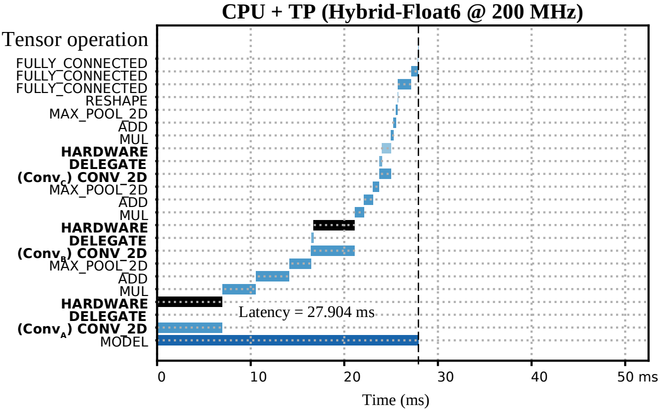
Runtime Profiling: Latencies and Scheduling on Floating-Point and Hybrid Float 6 Accelerators

Tensor Op	CPU (ms)	TP FP32 (ms)	Gain	**TP HF6 (ms)**	**Gain**
$Conv_A$	112.24	12.49	8.9	**6.92**	**16.2**
$Conv_B$	213.13	16.39	13.0		
$Conv_C$	46.59	3.59	12.9		



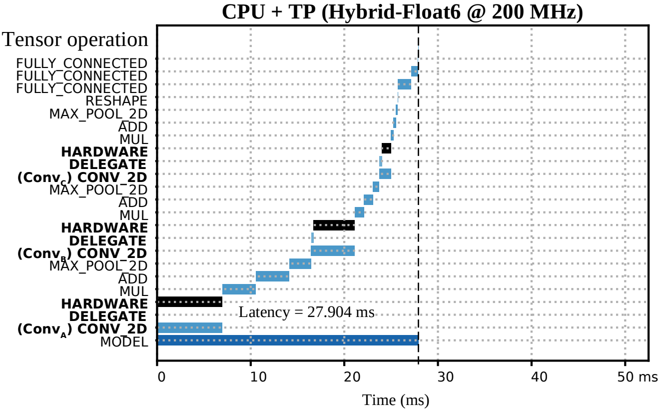
Runtime Profiling: Latencies and Scheduling on Floating-Point and Hybrid Float 6 Accelerators

Tensor Op	CPU (ms)	TP FP32 (ms)	Gain	**TP HF6 (ms)**	**Gain**
$Conv_A$	112.24	12.49	8.9	**6.92**	**16.2**
$Conv_B$	213.13	16.39	13.0	**4.41**	**48.3**
$Conv_C$	46.59	3.59	12.9		

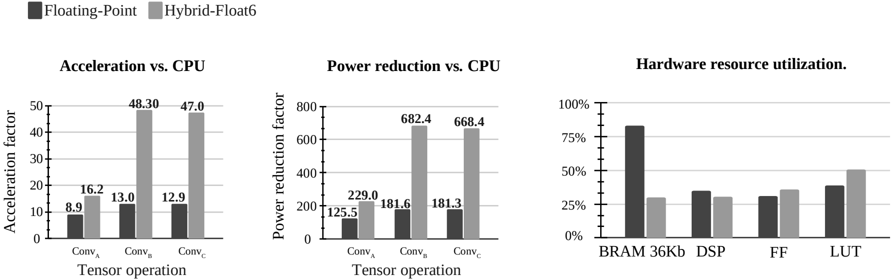


Runtime Profiling: Latencies and Scheduling on Floating-Point and Hybrid Float 6 Accelerators

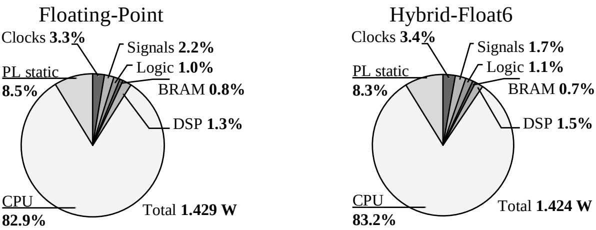
Tensor Op	CPU (ms)	TP FP32 (ms)	Gain	**TP HF6 (ms)**	**Gain**
$Conv_A$	112.24	12.49	8.9	**6.92**	**16.2**
$Conv_B$	213.13	16.39	13.0	**4.41**	**48.3**
$Conv_C$	46.59	3.59	12.9	**0.99**	**47.0**



Final Benchmarking Insights: Accelerators vs. CPU in Performance and Power Efficiency



Estimated power dissipation on the Zynq-7007S AP SoC with PS at 666 MHz and PL at 200 MHz



1. Methodology

2. Custom Floating-Point MAC Designs and Quantization Techniques

3. Case Studies

4. 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

- 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

- 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

- 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.

Thank You for Your Attention