



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

Yarib Nevarez

Universität Bremen

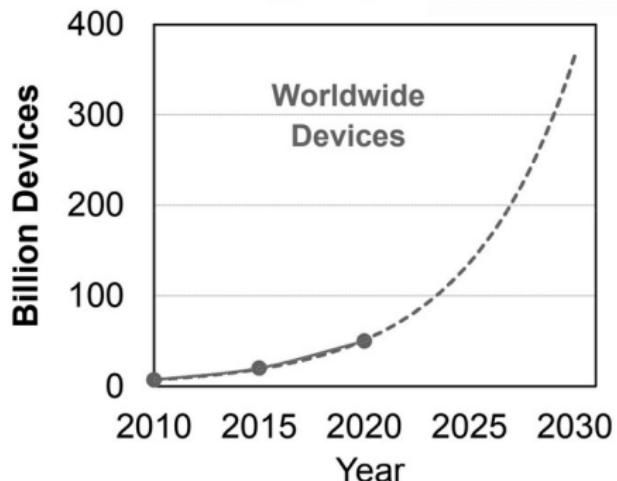
May 22, 2024

# 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

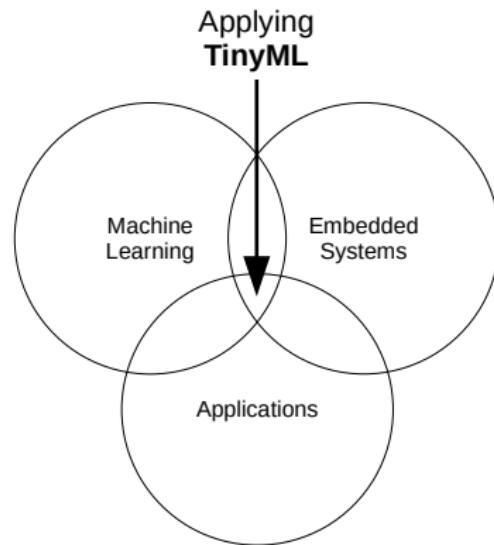


Loh, Kou-Hung Lawrence. "1.2 Fertilizing AIoT from roots to leaves." In 2020 IEEE International Solid-State Circuits Conference-(ISSCC), pp. 15-21. IEEE, 2020.

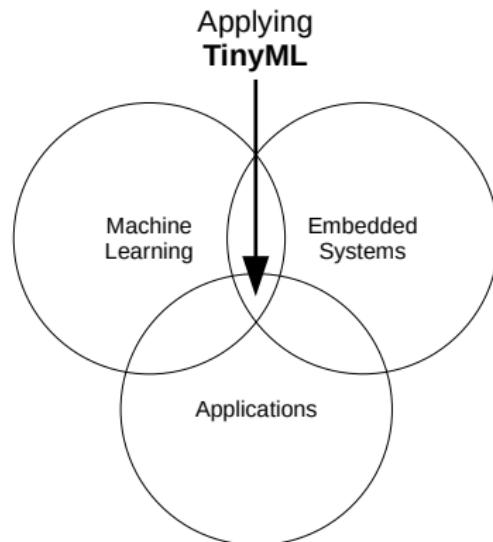
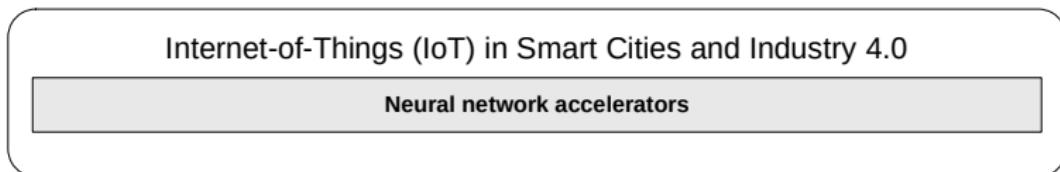
## Expansion of IoT and TinyML: Requirements, Accelerators, and Challenges

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



## 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

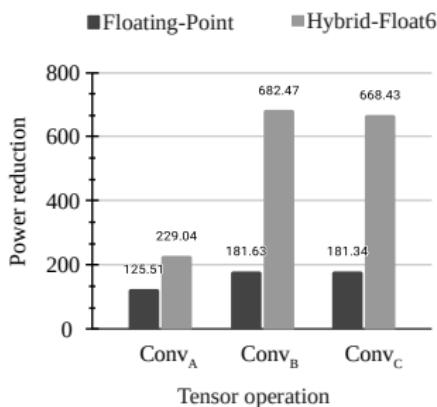
Quality preservation

Application versatility

Platform compatibility

On-device training

#### Power reduction relative to CPU



## 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:

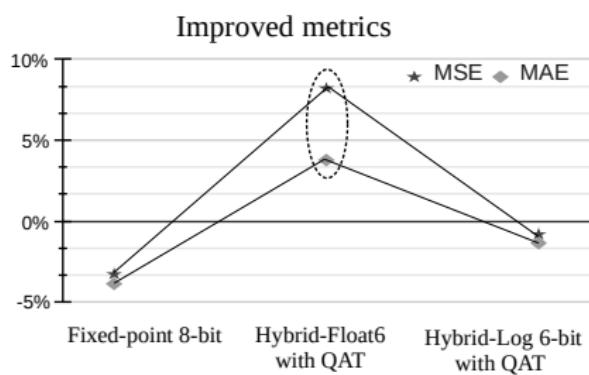
Energy and resource efficiency

**Quality preservation**

Application versatility

Platform compatibility

On-device training



# Introduction

## 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

#### Model versatility

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 ( $F \times T \times S$ )

## 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

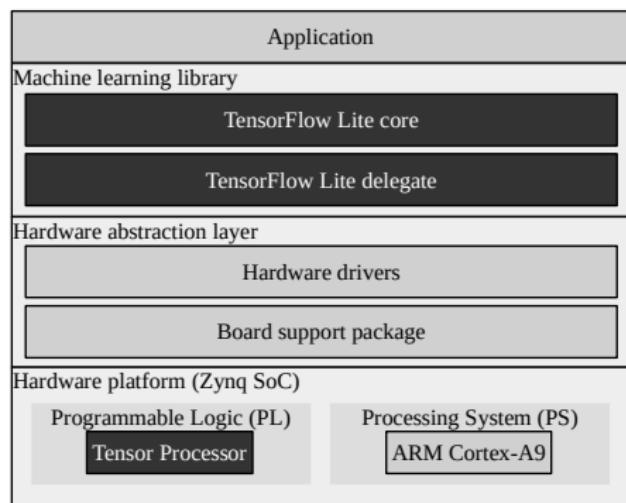
Quality preservation

Application versatility

**Platform compatibility**

On-device training

### HW/SW co-design framework



## 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:

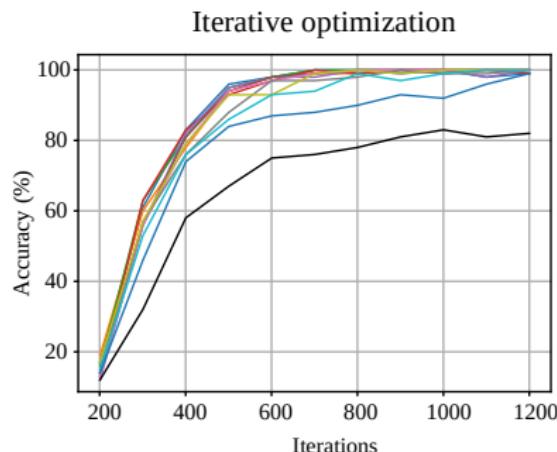
Energy and resource efficiency

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:

Current state-of-the-art methods:

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:

**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 future-proof neural network acceleration approach that supports inference and facilitates on-device training for TinyML applications.

## Goal and Objectives

- **Goal:** To establish a future-proof neural network acceleration approach that supports inference and facilitates on-device training for TinyML applications.
- **Objectives:**

## Goal and Objectives

- **Goal:** To establish a future-proof neural network acceleration approach that supports inference and facilitates on-device training for TinyML applications.
- **Objectives:**
  - Investigate optimizations for low-precision floating-point computation

## Goal and Objectives

- **Goal:** To establish a future-proof neural network acceleration approach that supports inference and facilitates on-device training for TinyML applications.
  
- **Objectives:**
  - Investigate optimizations for low-precision floating-point computation
  - Conduct design exploration

## Goal and Objectives

- **Goal:** To establish a future-proof neural network acceleration approach that supports inference and facilitates on-device training for TinyML applications.
- **Objectives:**
  - Investigate optimizations for low-precision floating-point computation
  - Conduct design exploration
  - Evaluate deployment, performance and impact

## Goal and Objectives

- **Goal:** To establish a future-proof neural network acceleration approach that supports inference and facilitates on-device training for TinyML applications.
- **Objectives:**
  - Investigate optimizations for low-precision floating-point computation
  - Conduct design exploration
  - Evaluate deployment, performance and impact
  - Ensure cross-platform compatibility

# Outline

- 1 Methodology
- 2 Custom Floating-Point MAC Designs and Quantization Techniques
- 3 Case Studies
- 4 Conclusions

1 Methodology

2 Custom Floating-Point MAC Designs and Quantization Techniques

3 Case Studies

4 Conclusions

# Methodology

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

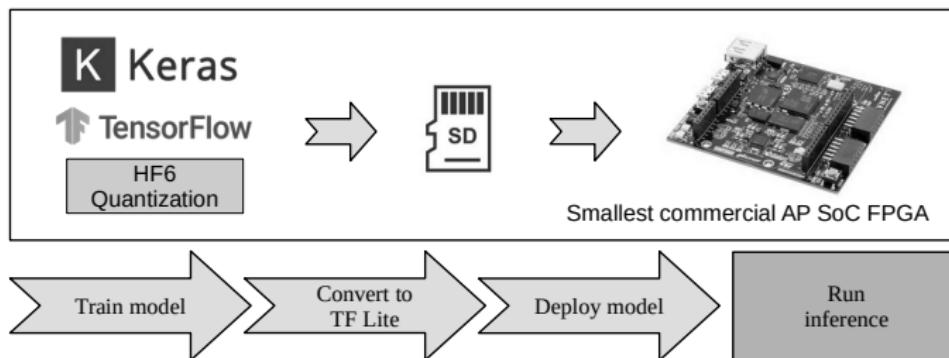
The methodology efficiently deploys and accelerates floating-point neural networks on embedded systems, optimizing performance, energy consumption, and hardware utilization.

Characteristics:

Streamlined hardware architecture

Custom floating-point arithmetic

Neural network execution without quantization processes



# Methodology

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

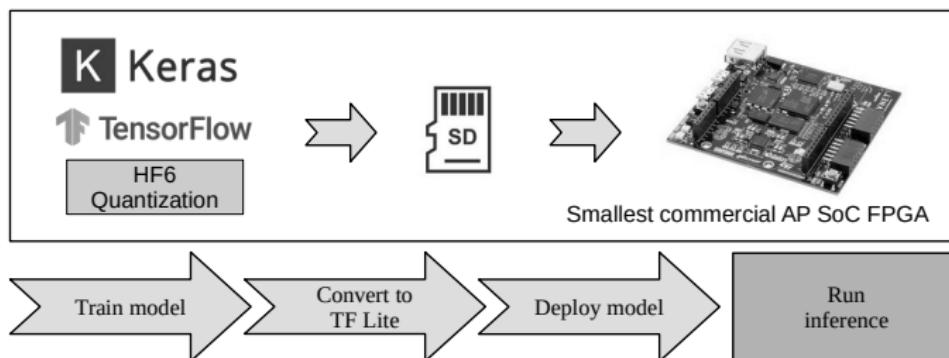
The methodology efficiently deploys and accelerates floating-point neural networks on embedded systems, optimizing performance, energy consumption, and hardware utilization.

### Characteristics:

Streamlined hardware architecture

Custom floating-point arithmetic

Neural network execution without quantization processes



# Methodology

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

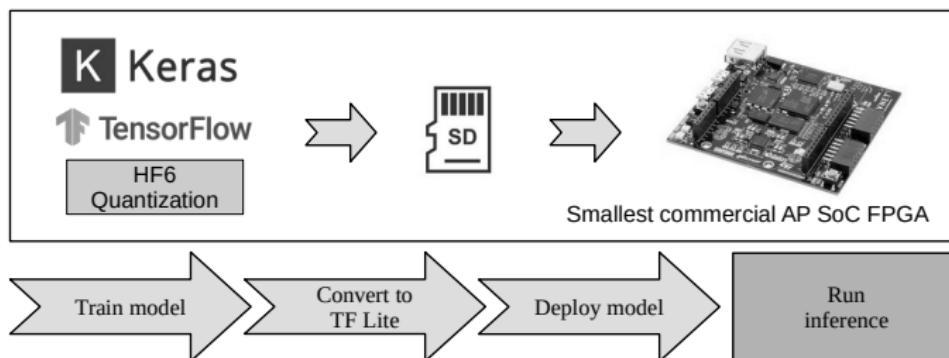
The methodology efficiently deploys and accelerates floating-point neural networks on embedded systems, optimizing performance, energy consumption, and hardware utilization.

### Characteristics:

Streamlined hardware architecture

Custom floating-point arithmetic

Neural network execution without quantization processes



# Methodology

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

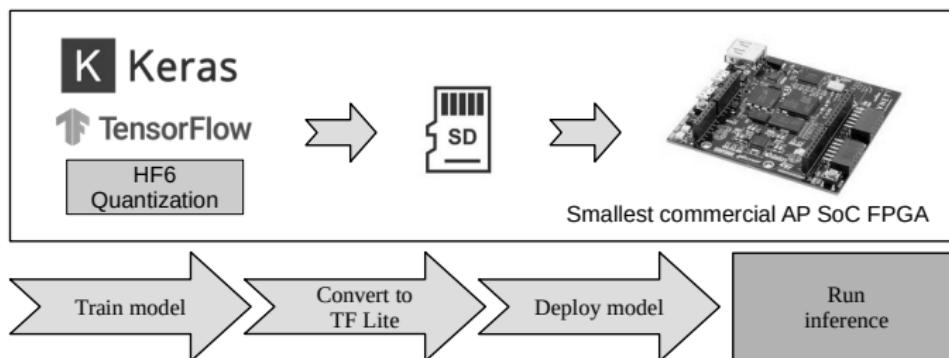
The methodology efficiently deploys and accelerates floating-point neural networks on embedded systems, optimizing performance, energy consumption, and hardware utilization.

### Characteristics:

Streamlined hardware architecture

Custom floating-point arithmetic

Neural network execution without quantization processes



# Methodology

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

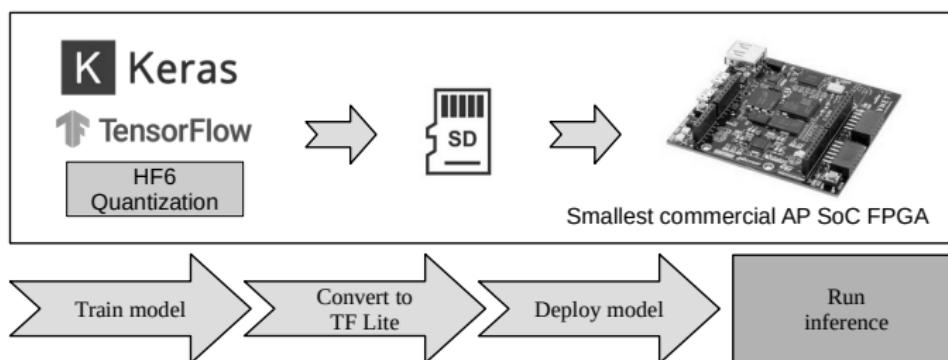
The methodology efficiently deploys and accelerates floating-point neural networks on embedded systems, optimizing performance, energy consumption, and hardware utilization.

### Characteristics:

Streamlined hardware architecture

Custom floating-point arithmetic

Neural network execution without quantization processes



## 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

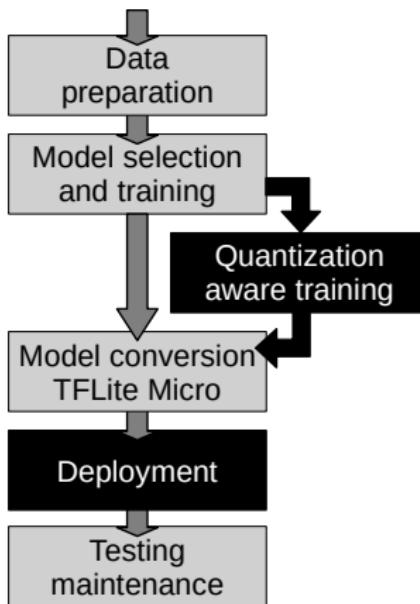
# Methodology

## 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



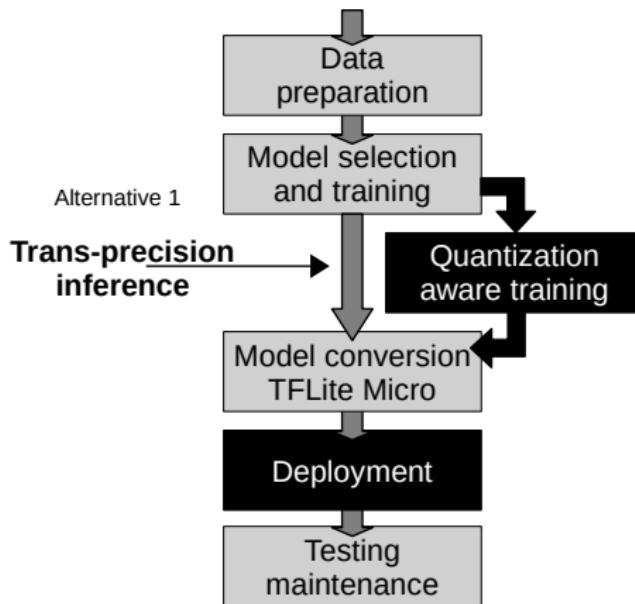
# Methodology

## 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



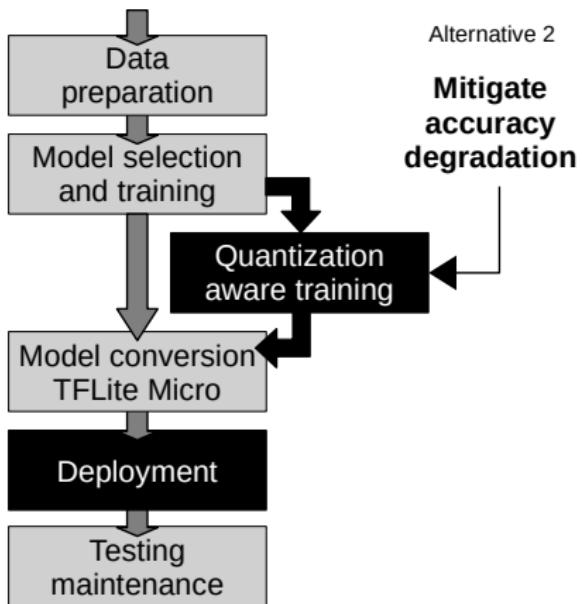
# Methodology

## 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



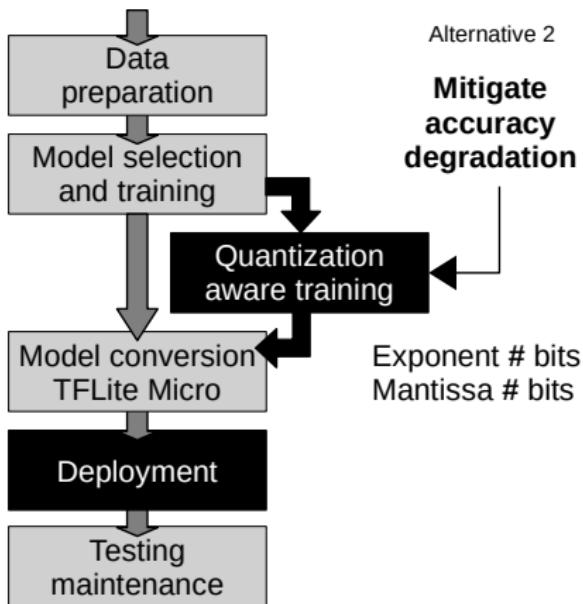
# Methodology

## 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



# 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:

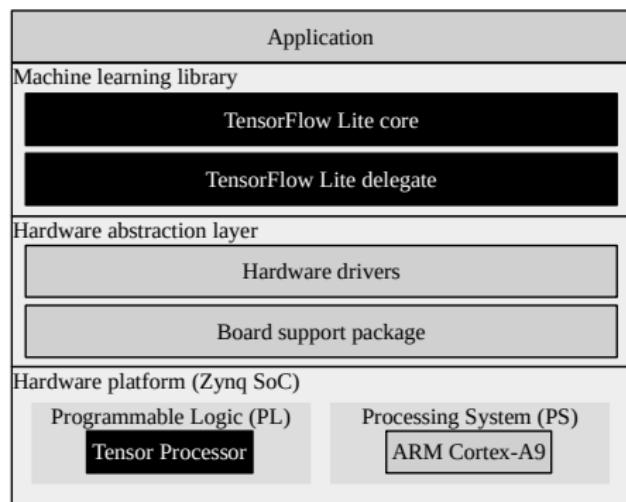
1. Model deployment

### 2. System infrastructure

3. Streamlined acceleration

4. Optimized processing

### 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

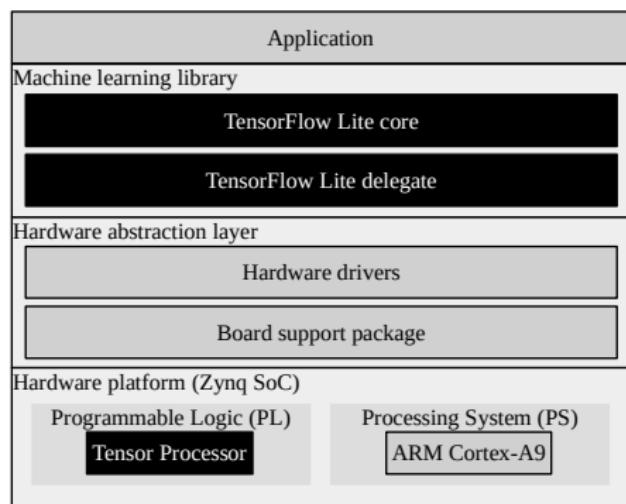
**2. System infrastructure**

3. Streamlined acceleration

4. Optimized processing

Industry standard framework

HW/SW co-design framework



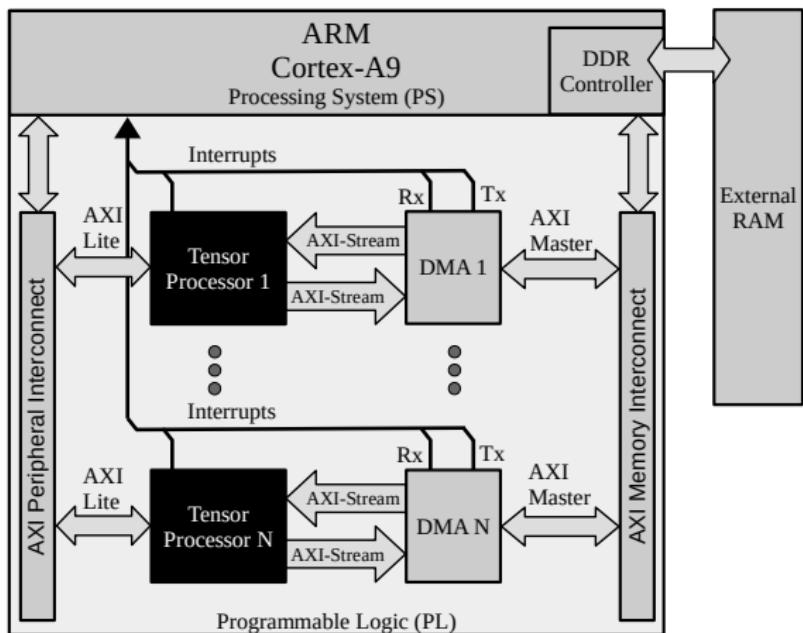
# Methodology

## 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



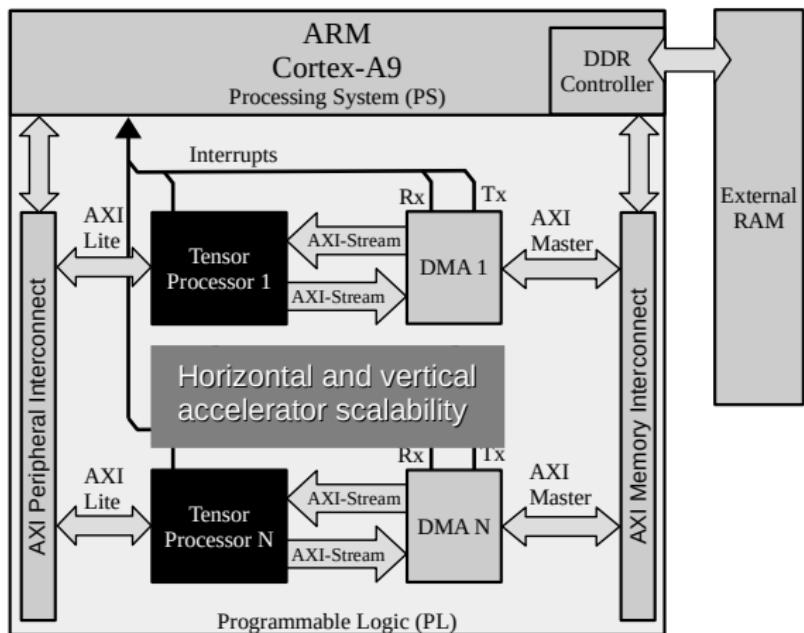
# Methodology

## 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



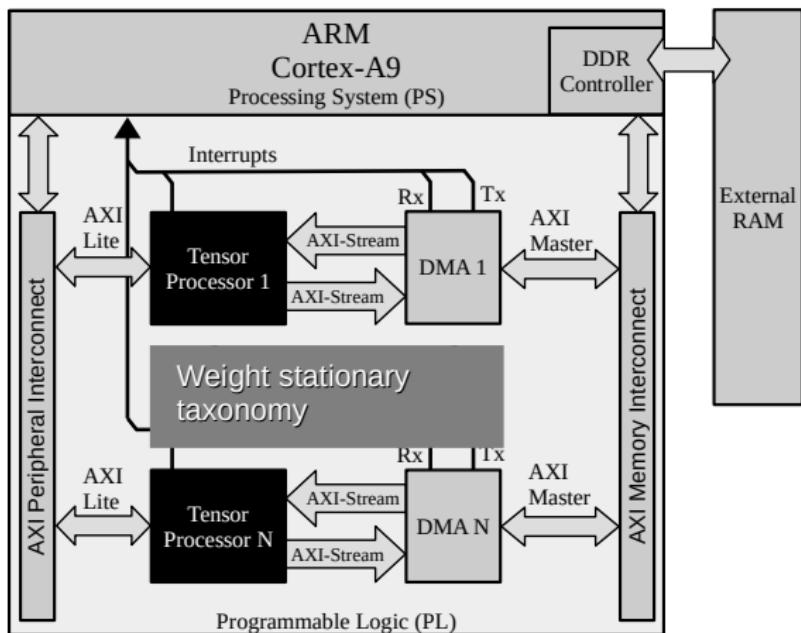
# Methodology

## 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



# 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.

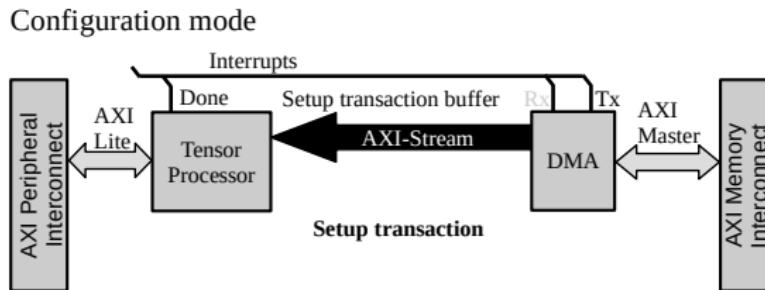
### Abstraction levels:

1. Model deployment

2. System infrastructure

### 3. Streamlined acceleration

4. Optimized processing



# 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.

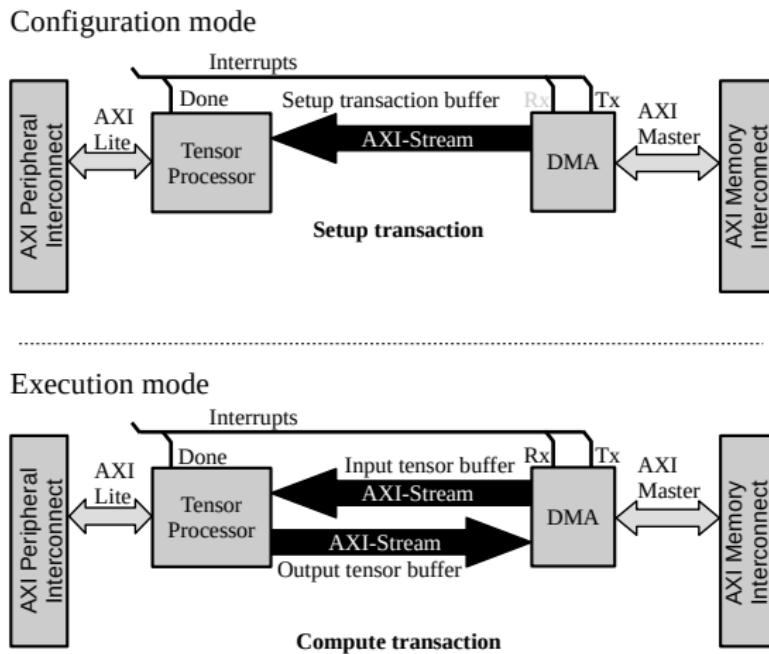
### Abstraction levels:

1. Model deployment

2. System infrastructure

### 3. Streamlined acceleration

4. Optimized processing



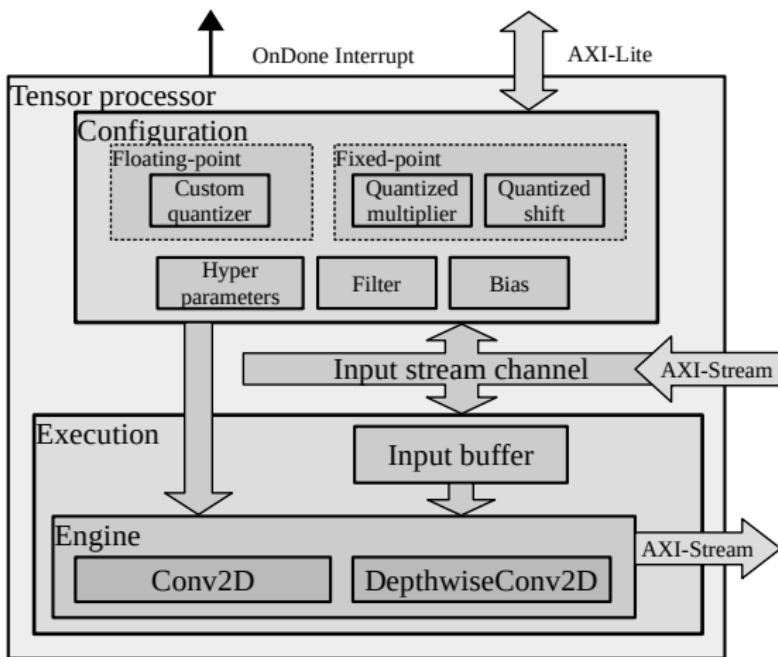
# Methodology

## 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



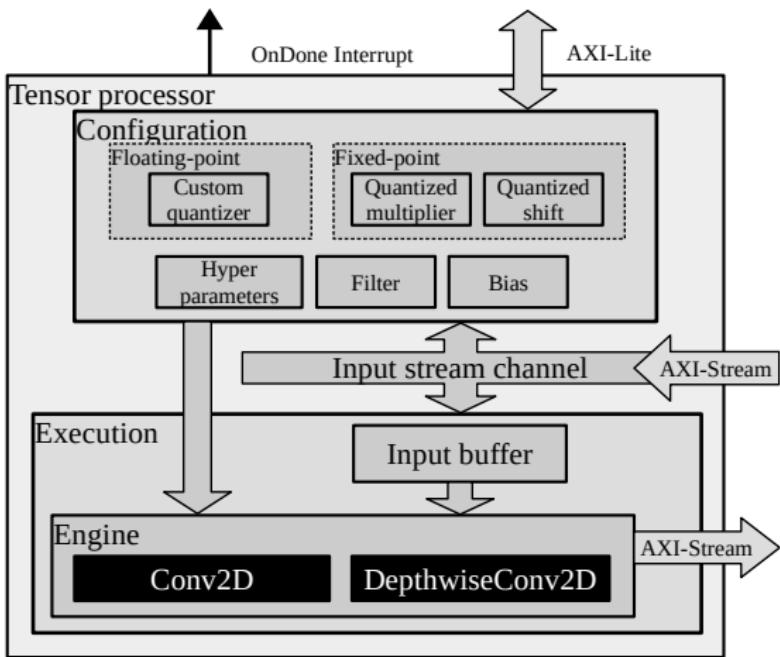
# Methodology

## 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



# Methodology

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

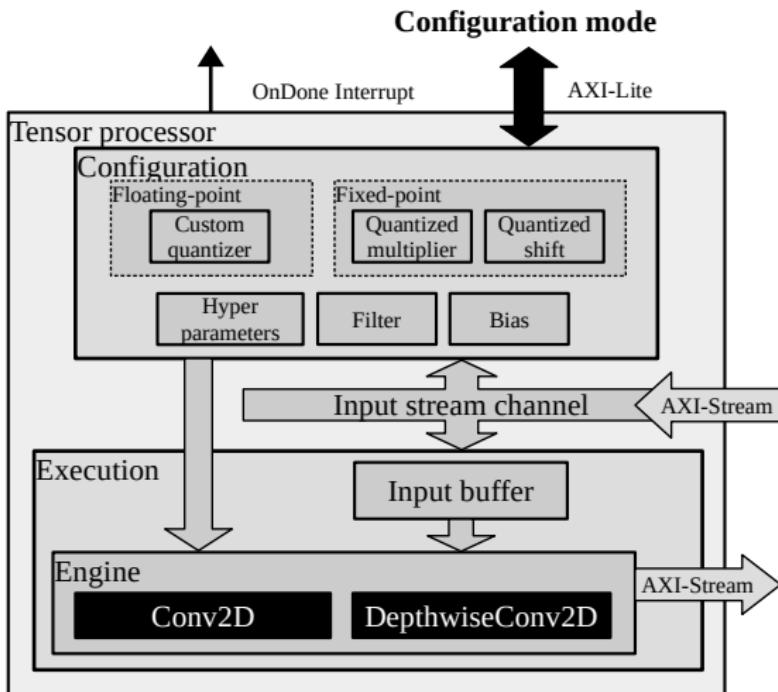
The methodology efficiently deploys and accelerates floating-point neural networks on embedded systems, optimizing performance, energy consumption, and hardware utilization.

### Abstraction levels:

1. Model deployment
2. System infrastructure

**3. Streamlined acceleration**

4. Optimized processing



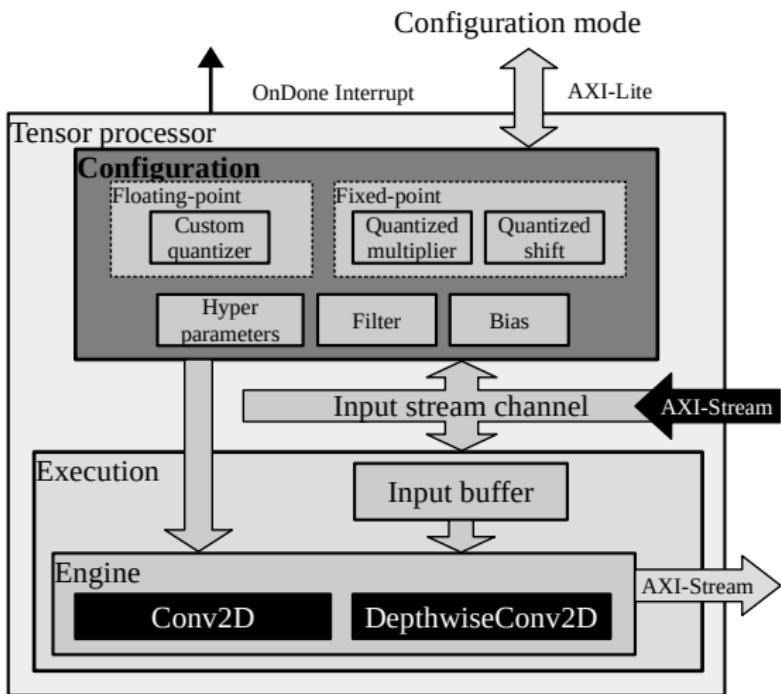
# Methodology

## 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



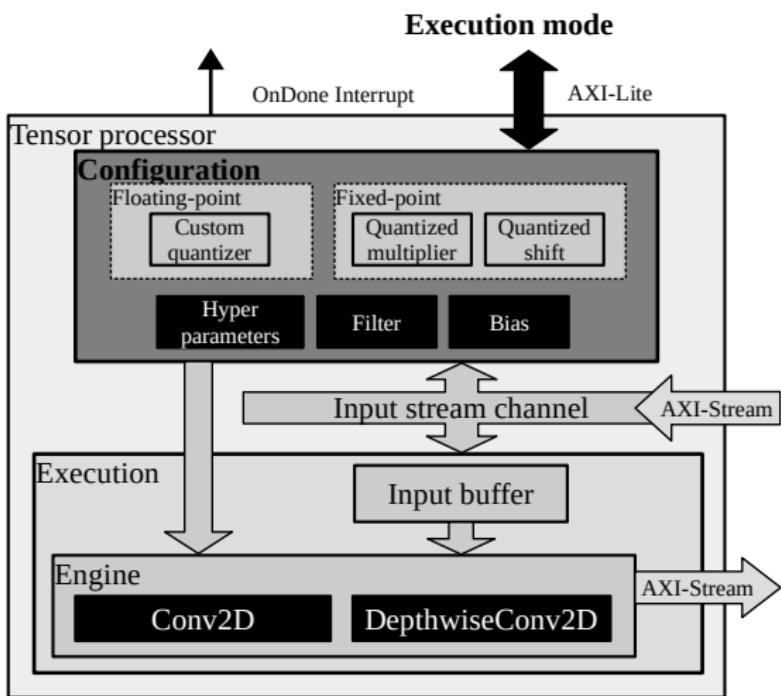
# Methodology

## 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



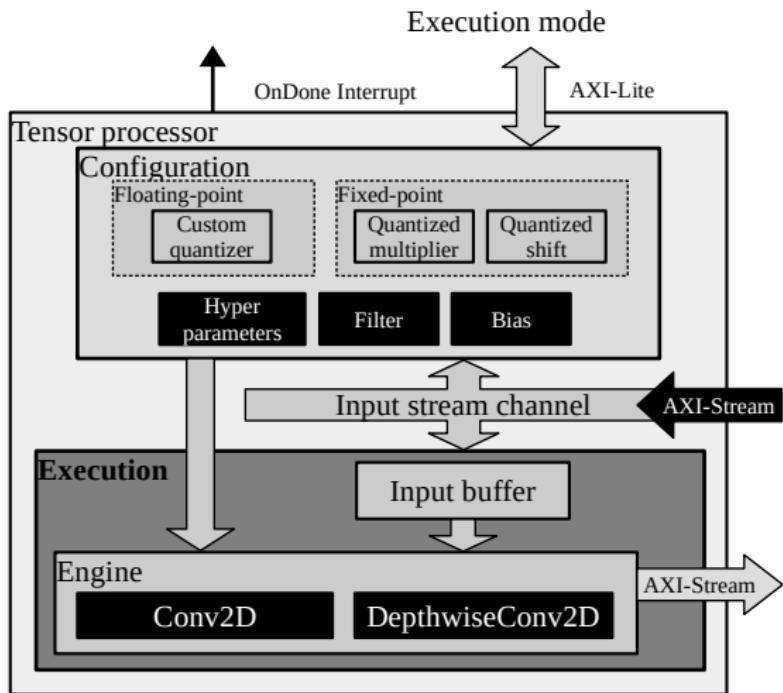
# Methodology

## 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



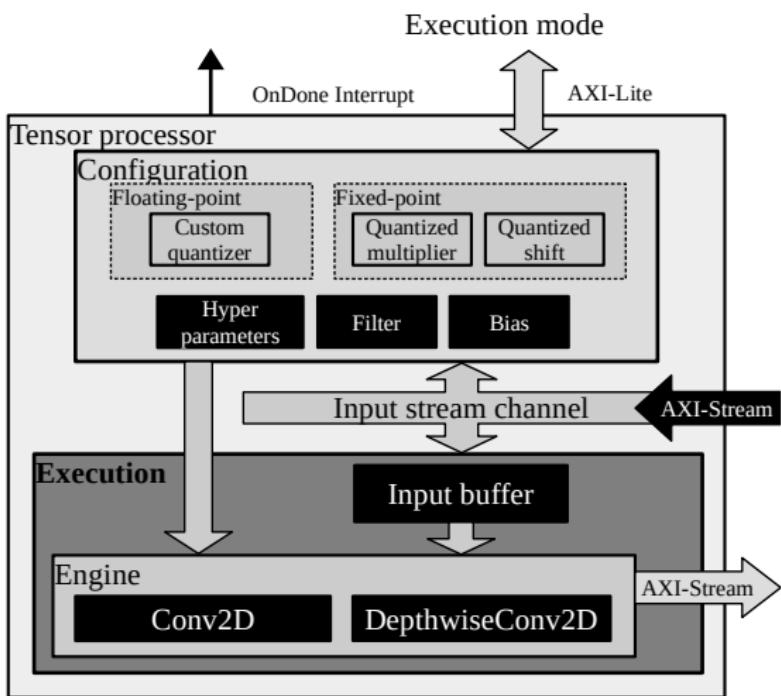
# Methodology

## 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



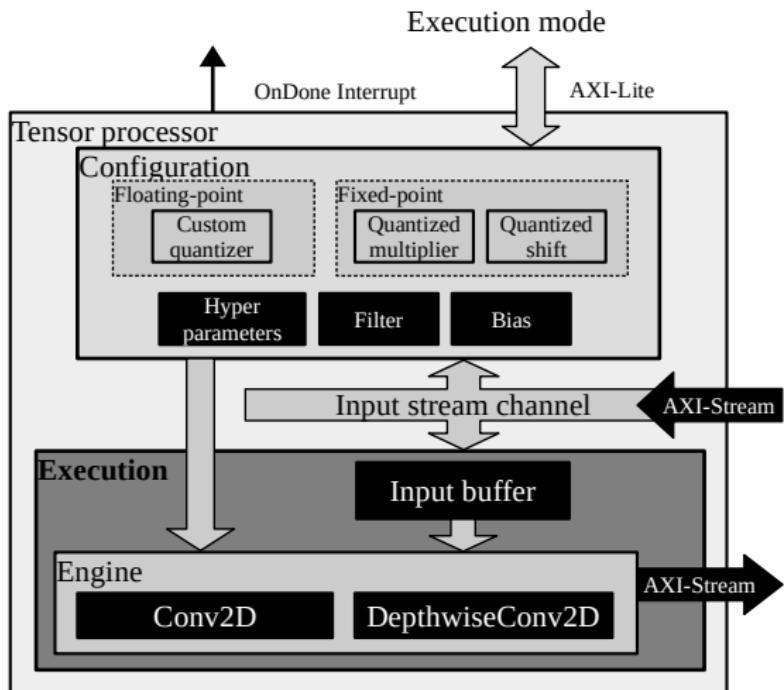
# Methodology

## 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



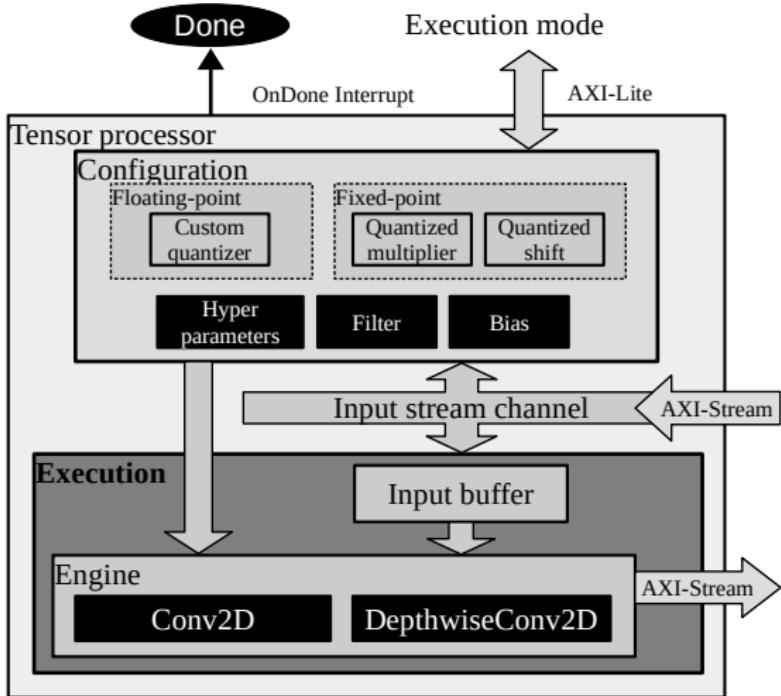
# Methodology

## 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



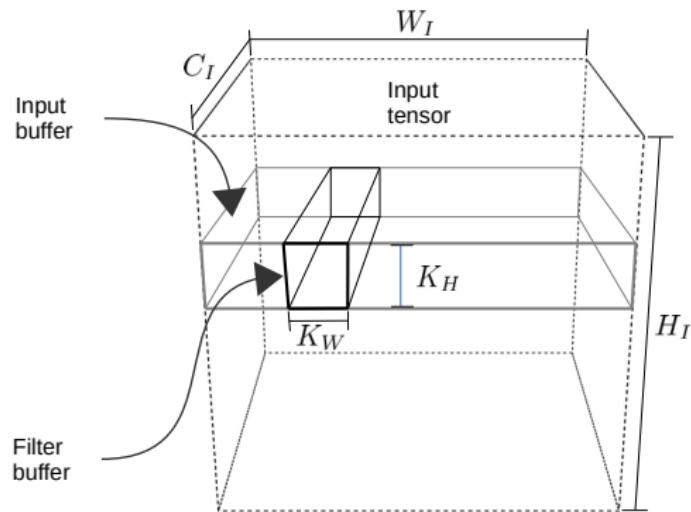
# Methodology

## 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



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

# Methodology

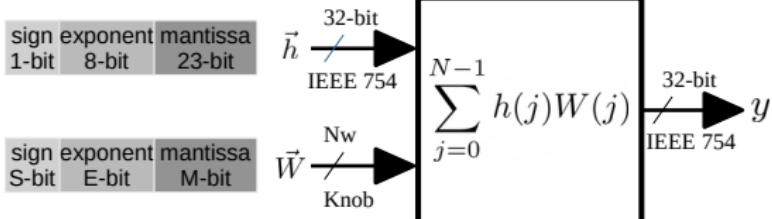
## Trans-Precision Neural Network Deployment for Low-Power Embedded Systems

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

### Abstraction levels:

Multiply-Accumulate Unit  
Hybrid Custom floating-point computation

1. Model deployment



2. System infrastructure

3. Streamlined acceleration

4. Optimized processing

# Methodology

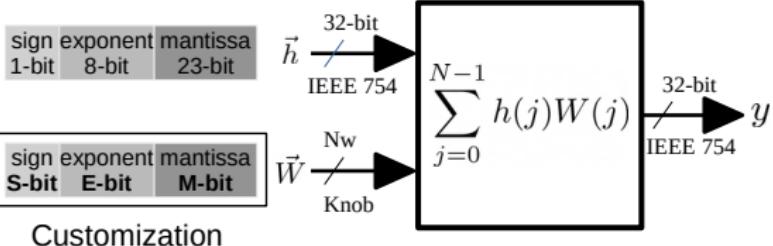
## Trans-Precision Neural Network Deployment for Low-Power Embedded Systems

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

### Abstraction levels:

Multiply-Accumulate Unit  
Hybrid Custom floating-point computation

1. Model deployment



2. System infrastructure

3. Streamlined acceleration

4. Optimized processing

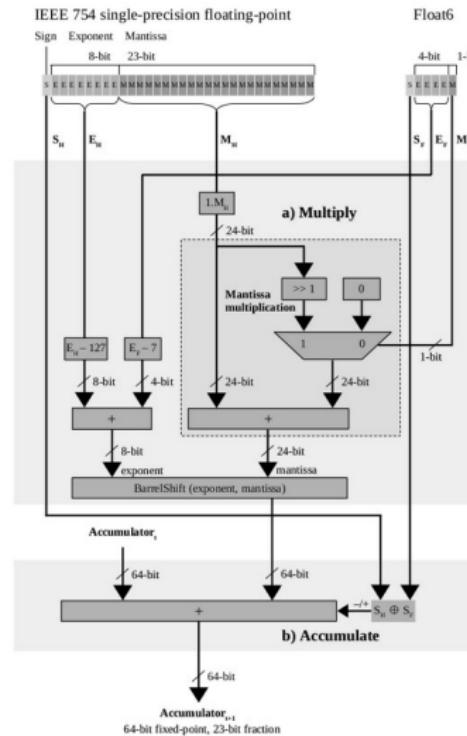
# Methodology

## 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



1 Methodology

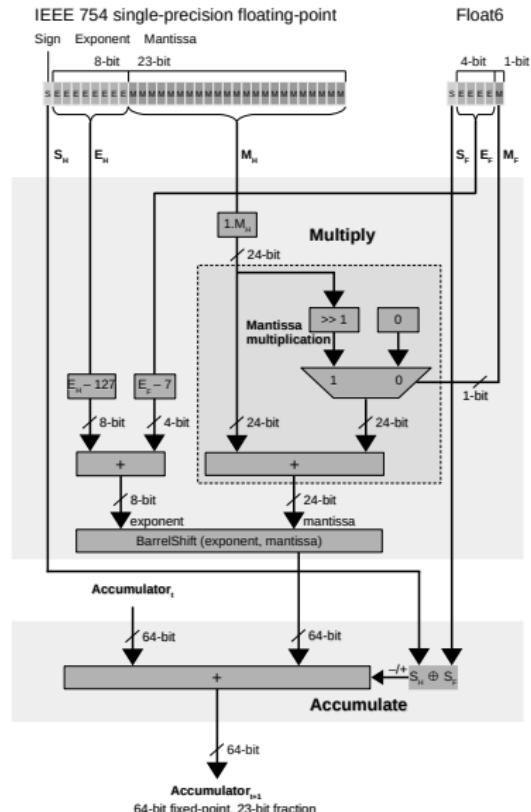
2 Custom Floating-Point MAC Designs and Quantization Techniques

3 Case Studies

4 Conclusions

# Custom Floating-Point MAC Designs

## Hybrid Custom Floating-Point Multiply-Accumulate Unit



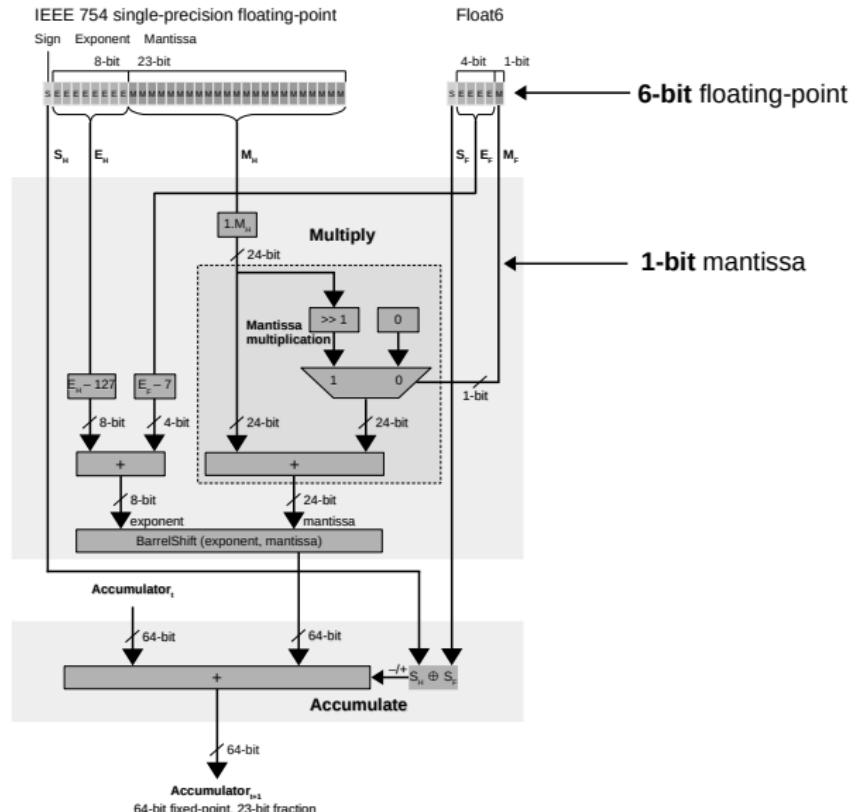
# Custom Floating-Point MAC Designs

## Hybrid Custom Floating-Point Multiply-Accumulate Unit

Vector dot-product:

Multiplication

Accumulation



# Custom Floating-Point MAC Designs

## Hybrid Custom Floating-Point Multiply-Accumulate Unit

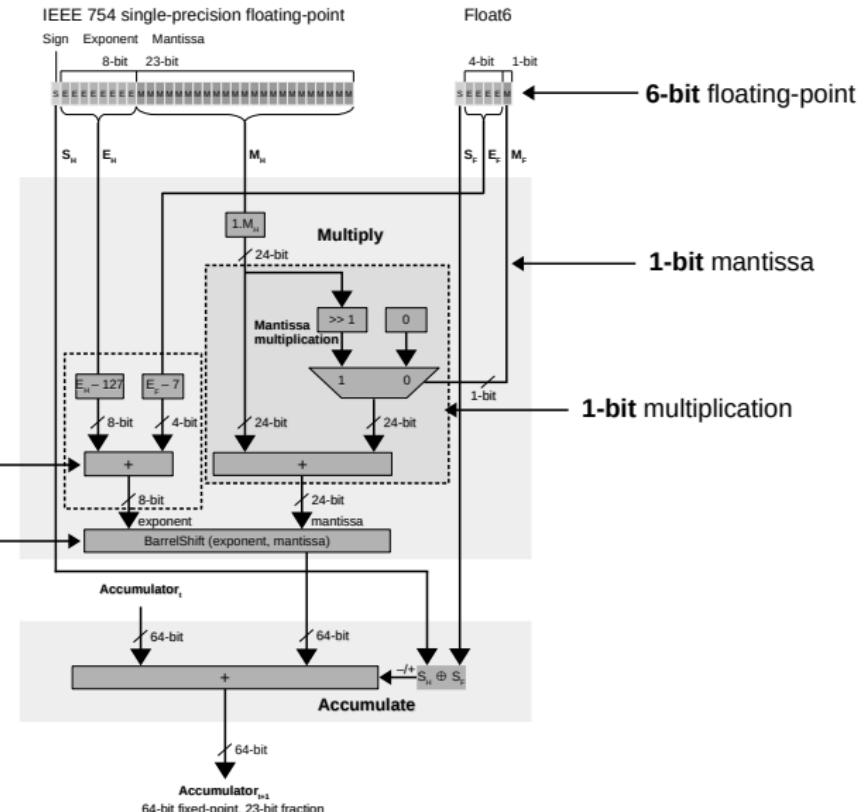
Vector dot-product:

Multiplication

Accumulation

Exponent addition

Denormalization



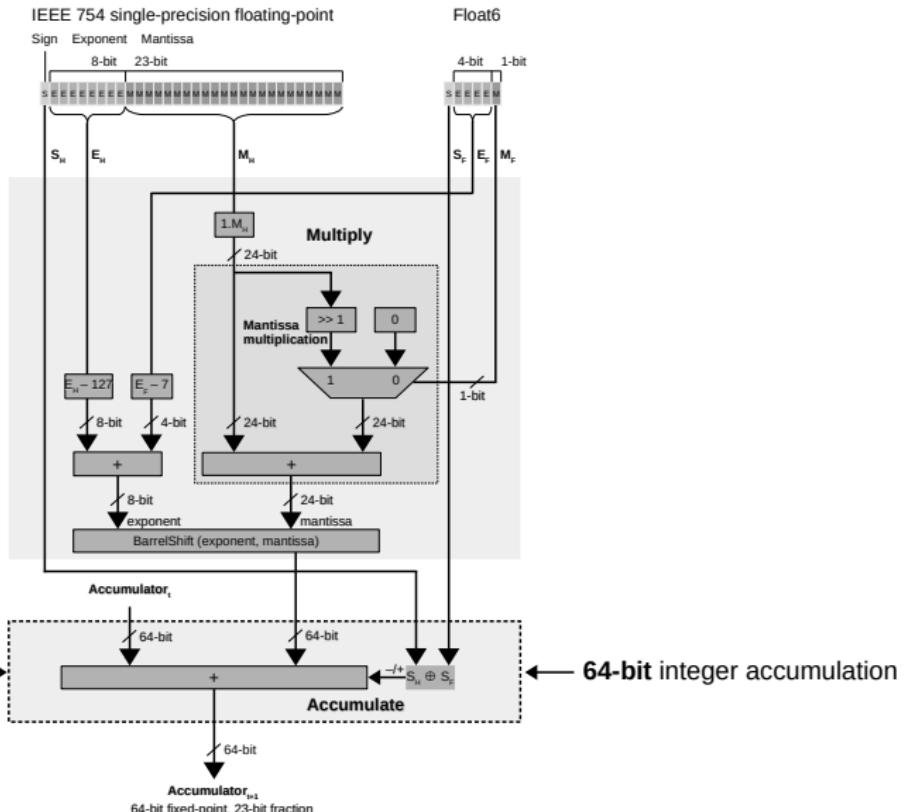
# Custom Floating-Point MAC Designs

## Hybrid Custom Floating-Point Multiply-Accumulate Unit

Vector dot-product:

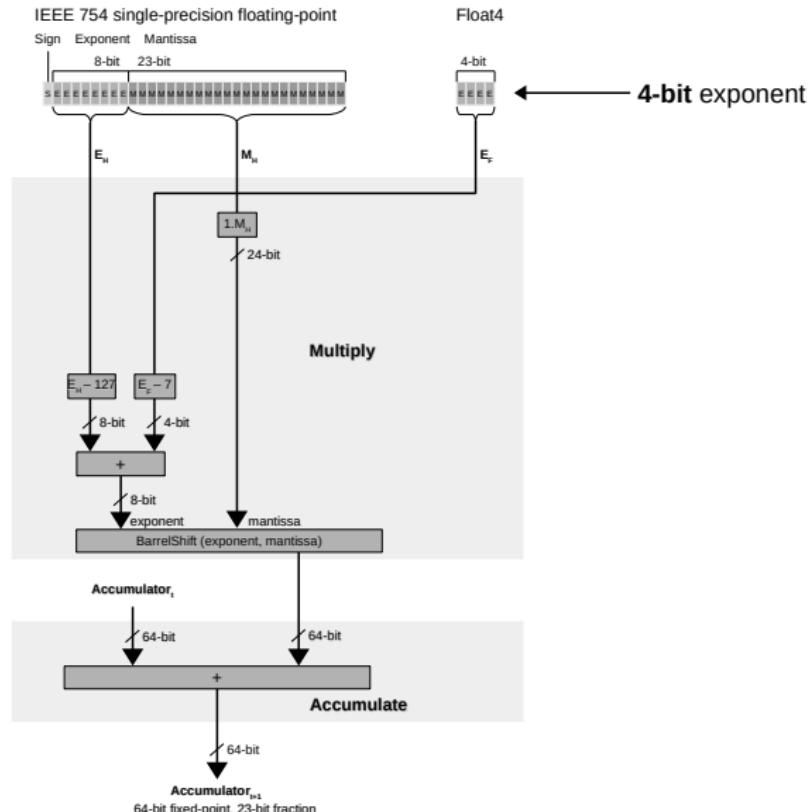
Multiplication

Accumulation



# Custom Floating-Point MAC Designs

## Hybrid Custom Floating-Point Multiply-Accumulate Unit



# Custom Floating-Point MAC Designs

## Hybrid Custom Floating-Point Multiply-Accumulate Unit

For noise-robust applications with  
non-negativity constraints.

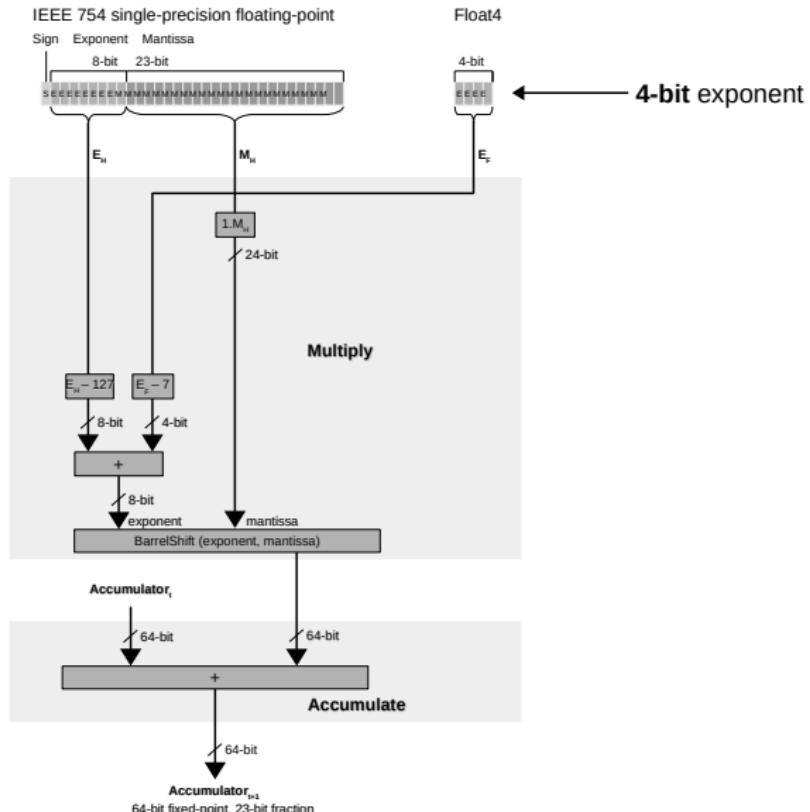
Non-negative Tensor Factorization

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: Esize as the target exponent bit size
Input: Msize as the target mantissa bits size
Input: STDMsize as the IEEE 754 mantissa bit size
Output: MODEL as the quantized CNN

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

---

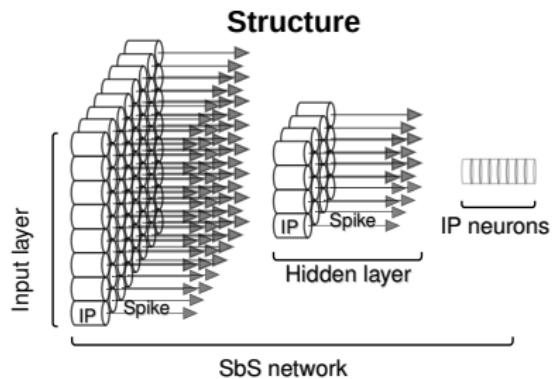
1 Methodology

2 Custom Floating-Point MAC Designs and Quantization Techniques

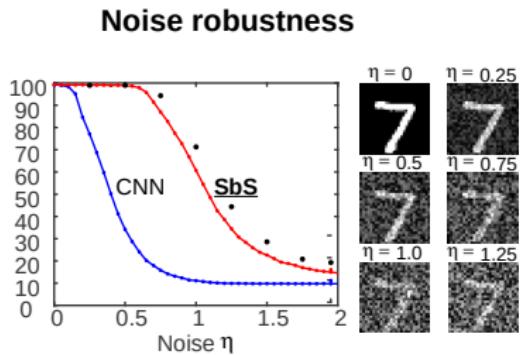
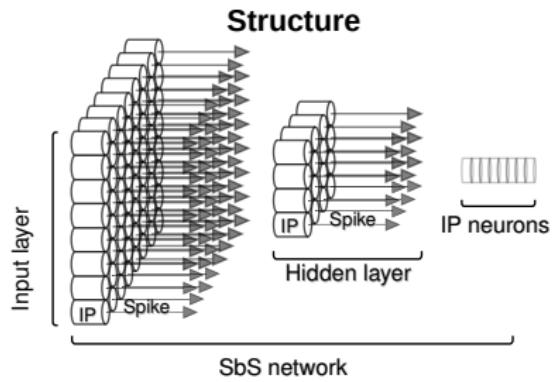
3 Case Studies

4 Conclusions

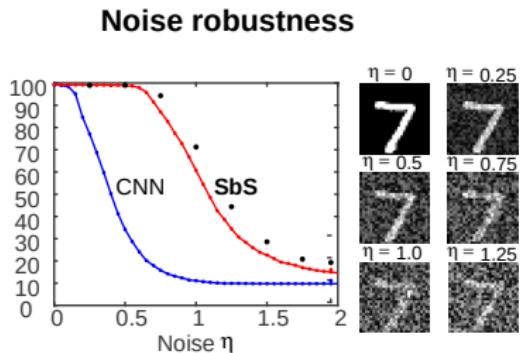
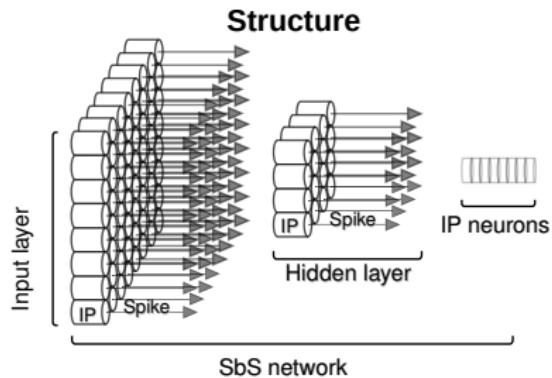
## Spike-by-Spike Neural Network



## Spike-by-Spike Neural Network



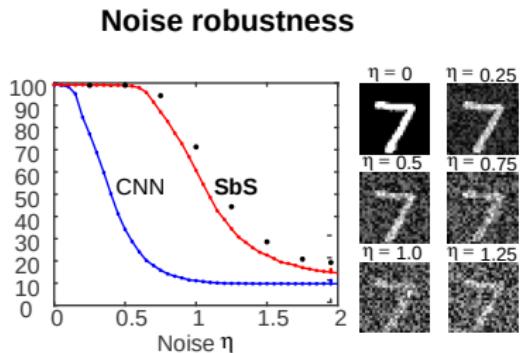
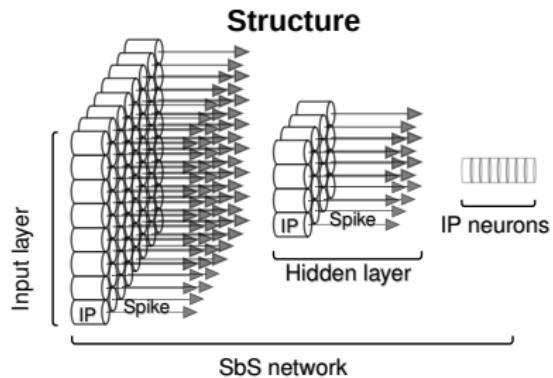
## Spike-by-Spike Neural Network



## 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

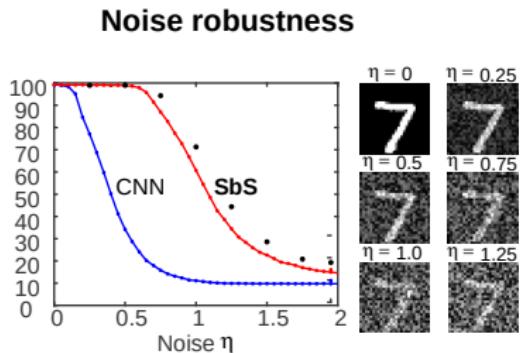
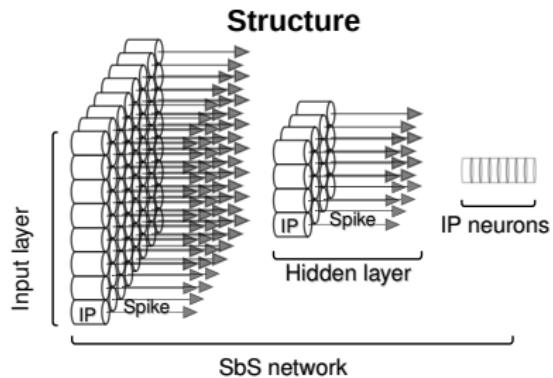


## Neuron update

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

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

## Spike-by-Spike Neural Network



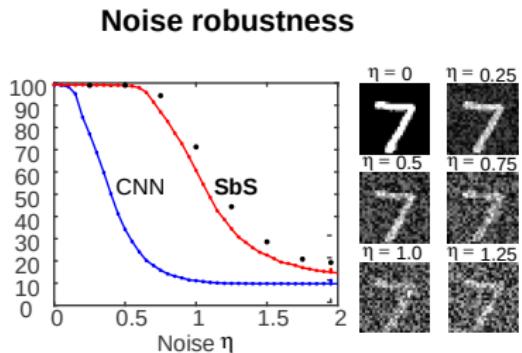
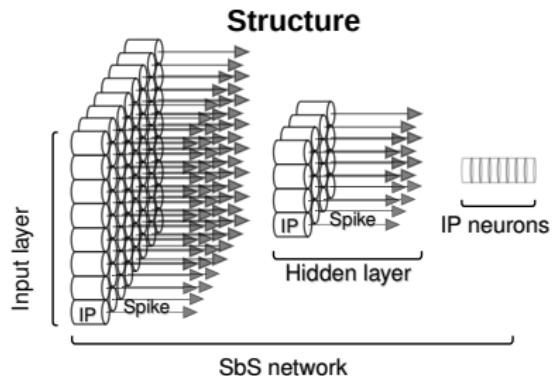
## Neuron update

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

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



### Properties:

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

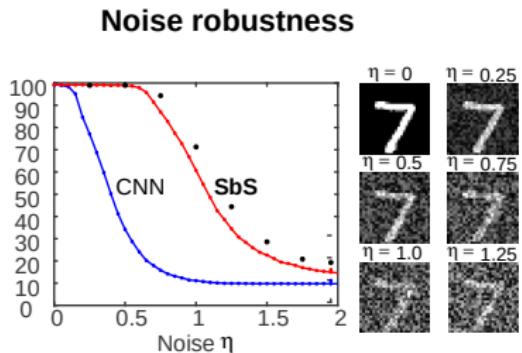
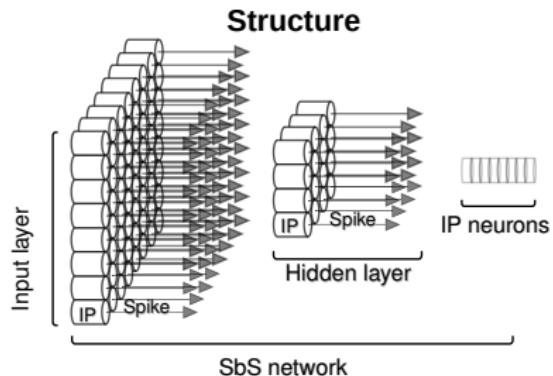
### Neuron update

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

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



### Properties:

#### Noise robustness

Iterative optimization

No sign bit required

Requires division

Compute and memory intensive

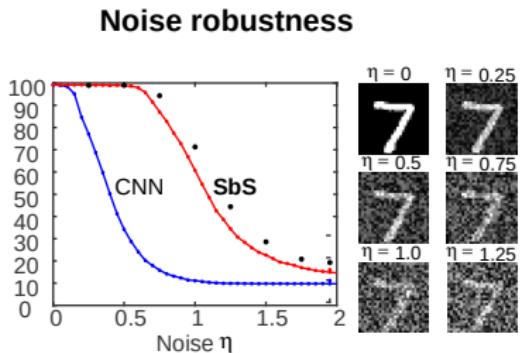
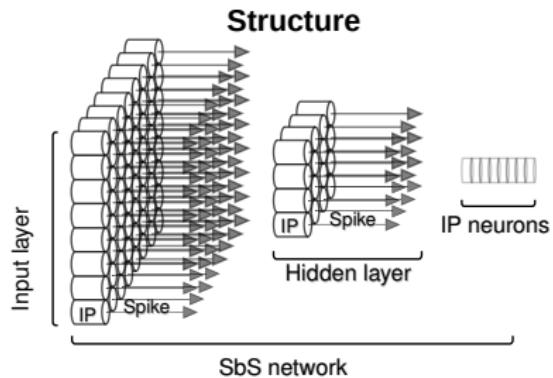
#### Neuron update

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

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



### Properties:

Noise robustness

### Iterative optimization

No sign bit required

Requires division

Compute and memory intensive

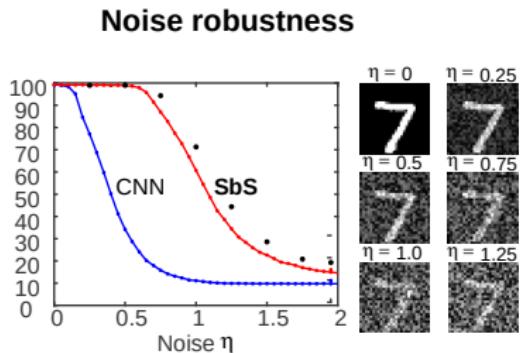
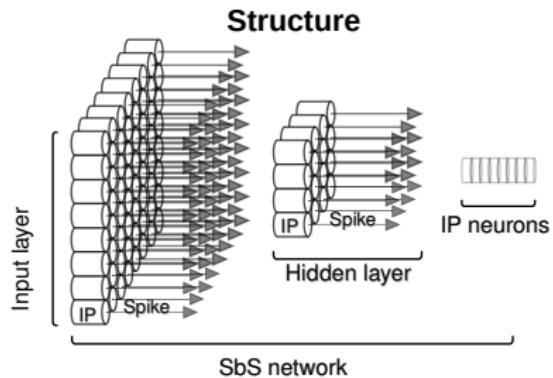
### Neuron update

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

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



### Properties:

Noise robustness

Iterative optimization

**No sign bit required**

Requires division

Compute and memory intensive

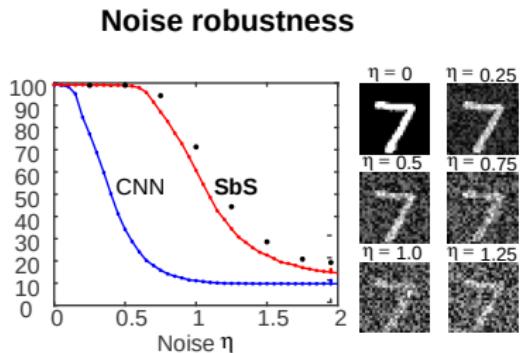
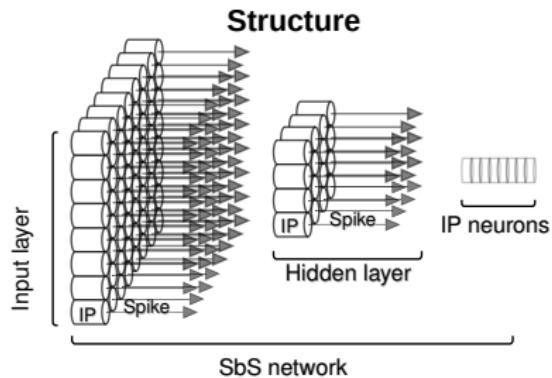
### Neuron update

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

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



## Properties:

## Noise robustness

## Iterative optimization

No sign bit required

### Requires division

Compute and memory intensive

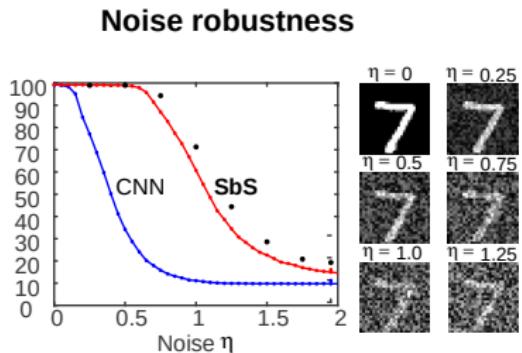
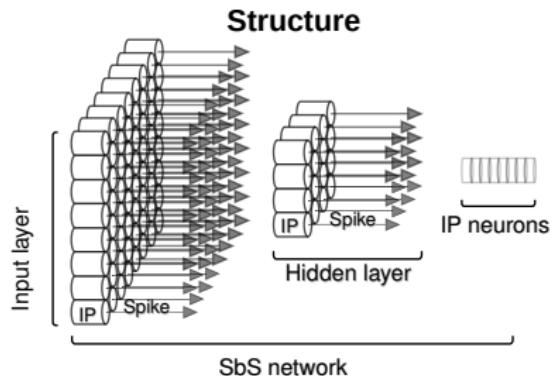
## Neuron update

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

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

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

## Spike-by-Spike Neural Network



### Properties:

Noise robustness

Iterative optimization

No sign bit required

Requires division

**Compute and memory intensive**

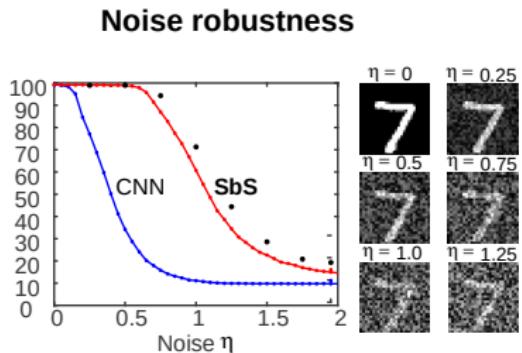
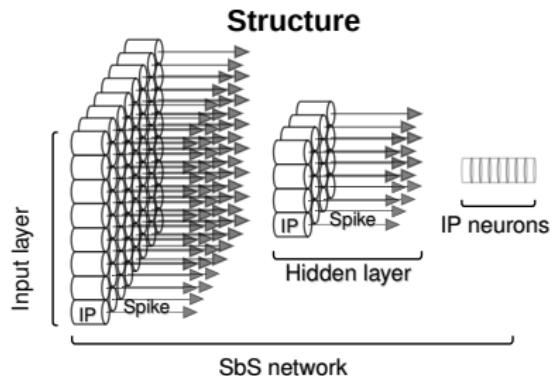
### Neuron update

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

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



### Properties:

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

### Neuron update

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

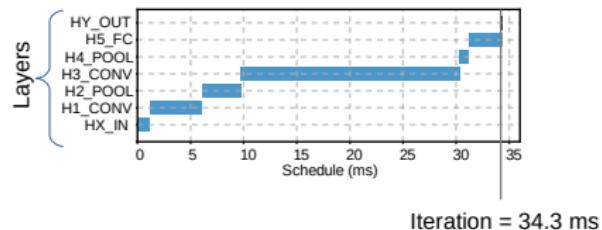
Non-negativity       $0 \leq W(s_t|j) \leq 1$

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

## Deployment

### Floating-point 32-bit

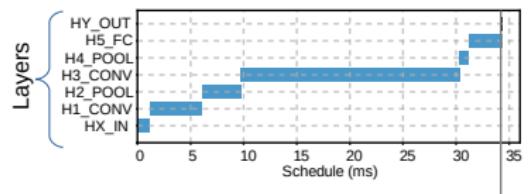
ARM Cortex A9 @ 666 MHz



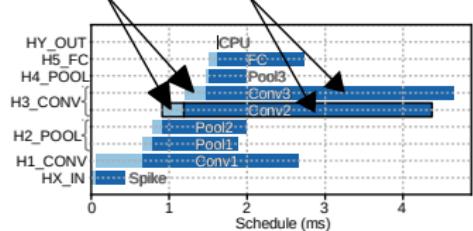
## Deployment

### Floating-point 32-bit

ARM Cortex A9 @ 666 MHz



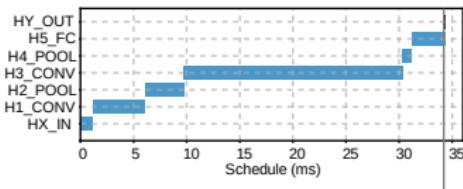
CPU      Accelerator      Iteration = 34.3 ms



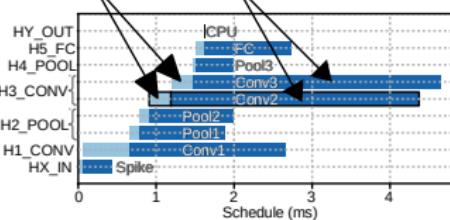
## Deployment

### Floating-point 32-bit

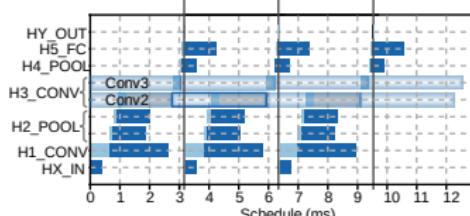
ARM Cortex A9 @ 666 MHz



CPU      Accelerator      Iteration = 34.3 ms



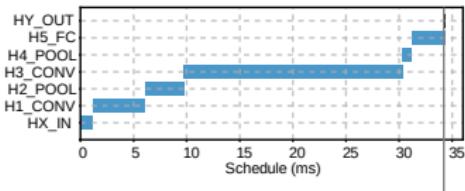
Iteration = 3.18 ms



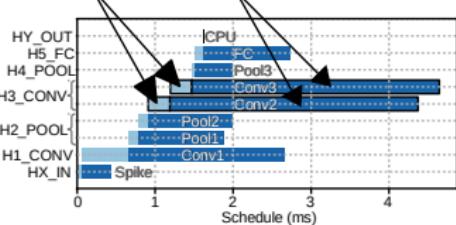
## Deployment

### Floating-point 32-bit

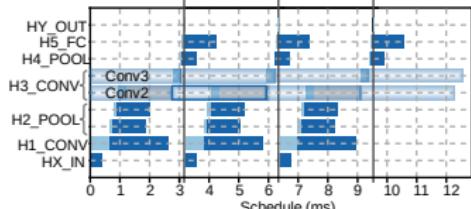
ARM Cortex A9 @ 666 MHz



CPU      Accelerator      **Iteration = 34.3 ms**



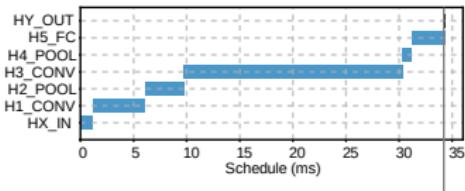
**Iteration = 3.18 ms**



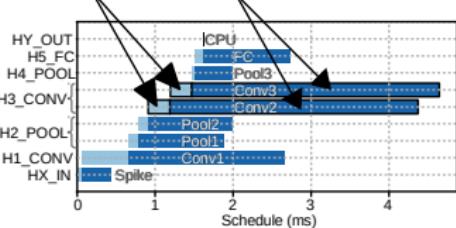
## Deployment

### Floating-point 32-bit

ARM Cortex A9 @ 666 MHz

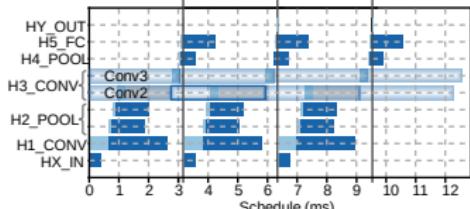


CPU      Accelerator      Iteration = 34.3 ms



Acceleration = 10.7

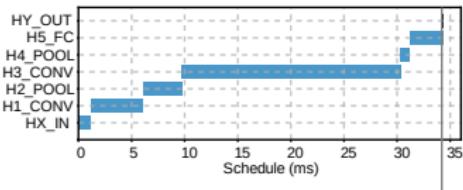
Iteration = 3.18 ms



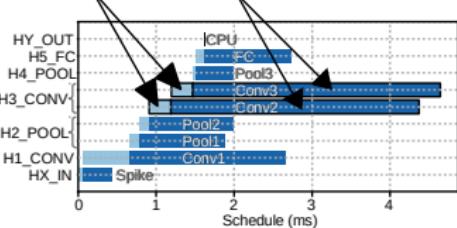
## Deployment

### Floating-point 32-bit

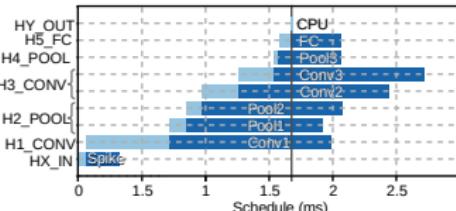
ARM Cortex A9 @ 666 MHz



CPU      Accelerator      Iteration = 34.3 ms

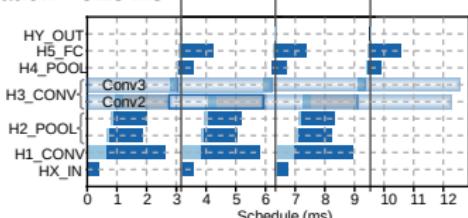


### Hybrid logarithmic 4-bit

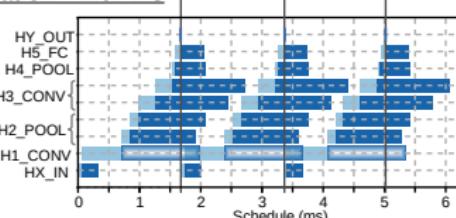


Acceleration = 10.7

Iteration = 3.18 ms



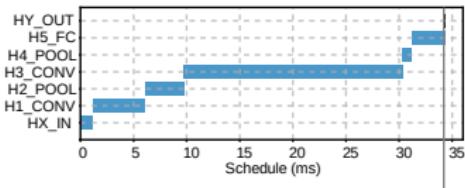
Iteration = 1.67 ms



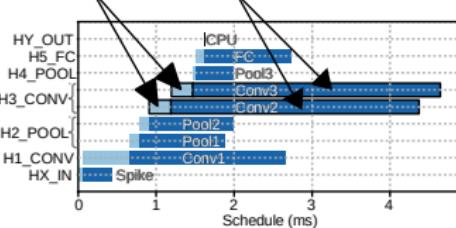
## Deployment

### Floating-point 32-bit

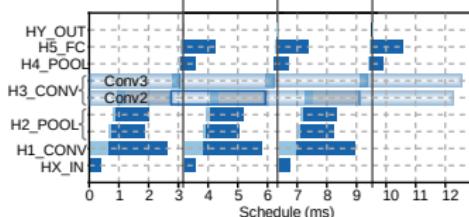
ARM Cortex A9 @ 666 MHz



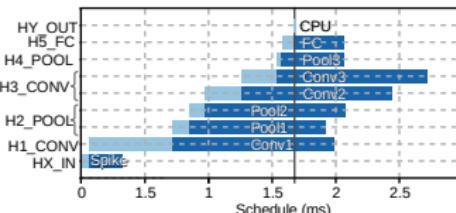
CPU Accelerator Iteration = 34.3 ms



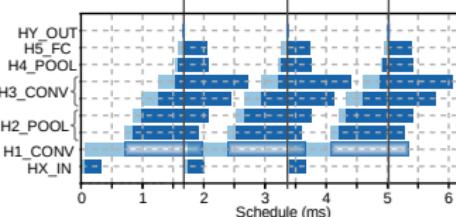
Acceleration = 10.7 Iteration = 3.18 ms



### Hybrid logarithmic 4-bit

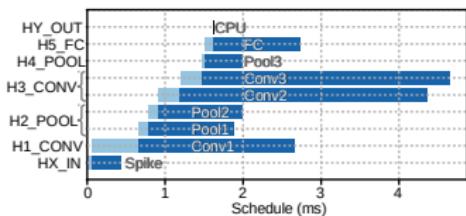


Acceleration = 20.5 Iteration = 1.67 ms

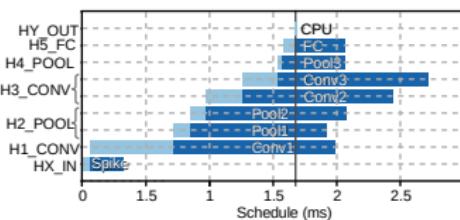


## Deployment

**Standard floating-point**

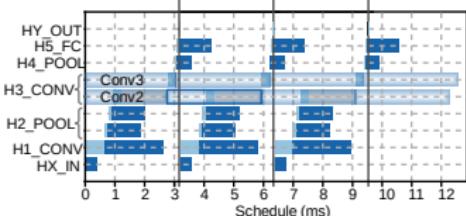


**Hybrid logarithmic 4-bit**



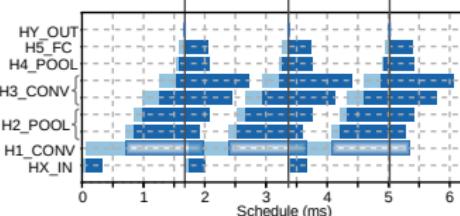
Iteration = 3.18 ms

**Acceleration = 10.7**



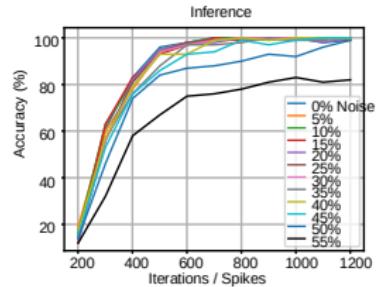
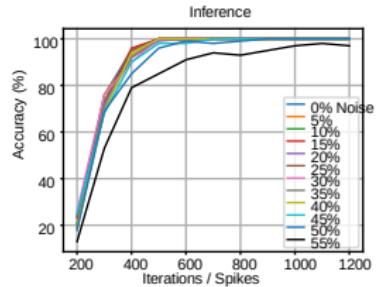
Iteration = 1.67 ms

**Acceleration = 20.5**

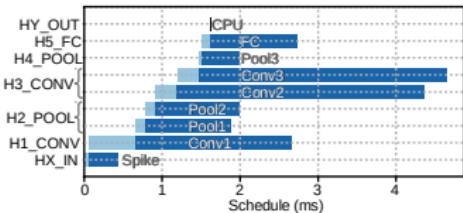


# Accelerating Spike-by-Spike Neural Networks with Hybrid Logarithmic 4-bit

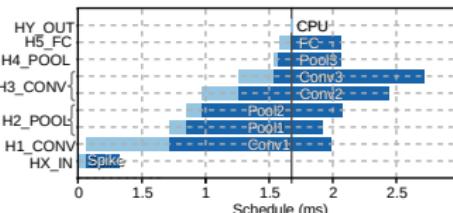
## Deployment



## Standard floating-point

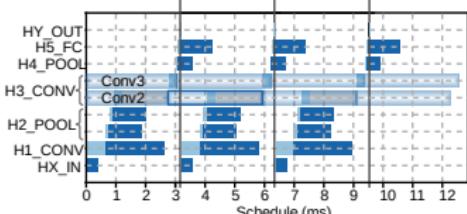


## Hybrid logarithmic 4-bit



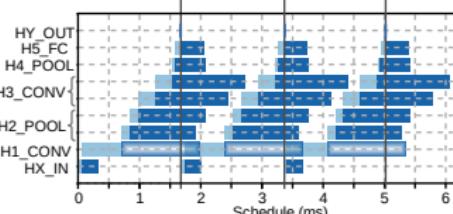
Acceleration = 10.7

Iteration = 3.18 ms



Acceleration = 20.5

Iteration = 1.67 ms



# Boosting TinyML Performance with Hybrid Floating-Point 6-bit

## TinyML Application: CNN Sensor Analytics for Structural Health Monitoring

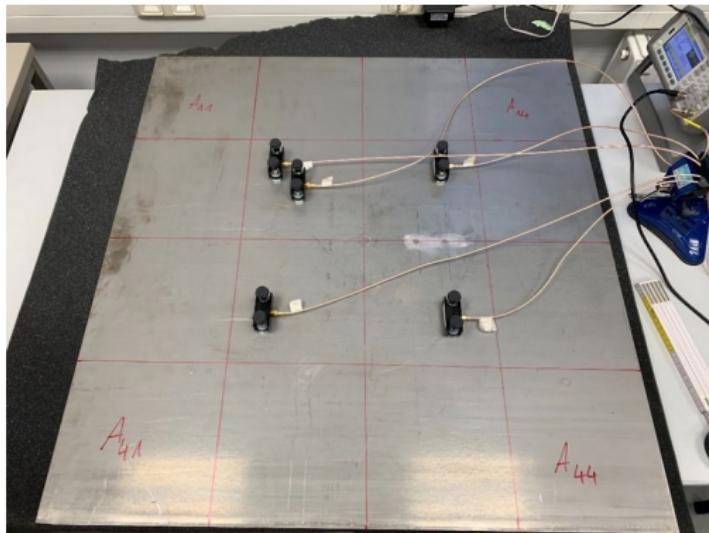
### Experimental setup:

Sensor and noise positions

Training dataset

Testing dataset

Model architecture



# Boosting TinyML Performance with Hybrid Floating-Point 6-bit

## TinyML Application: CNN Sensor Analytics for Structural Health Monitoring

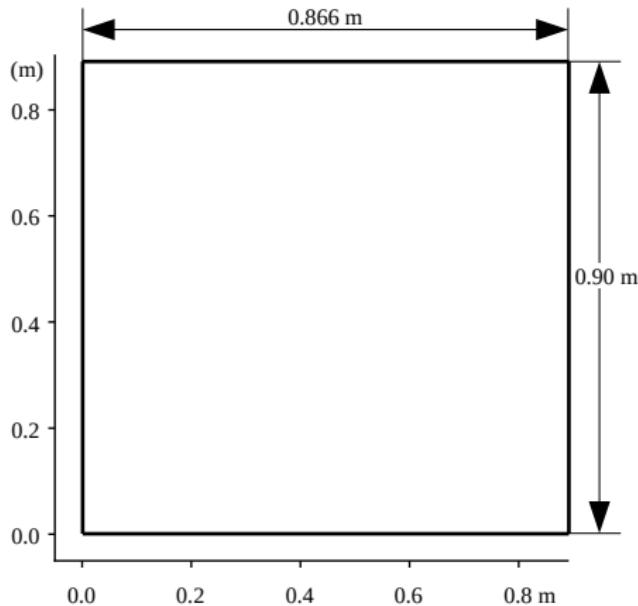
### Experimental setup:

Sensor and noise positions

Training dataset

Testing dataset

Model architecture



# Boosting TinyML Performance with Hybrid Floating-Point 6-bit

## TinyML Application: CNN Sensor Analytics for Structural Health Monitoring

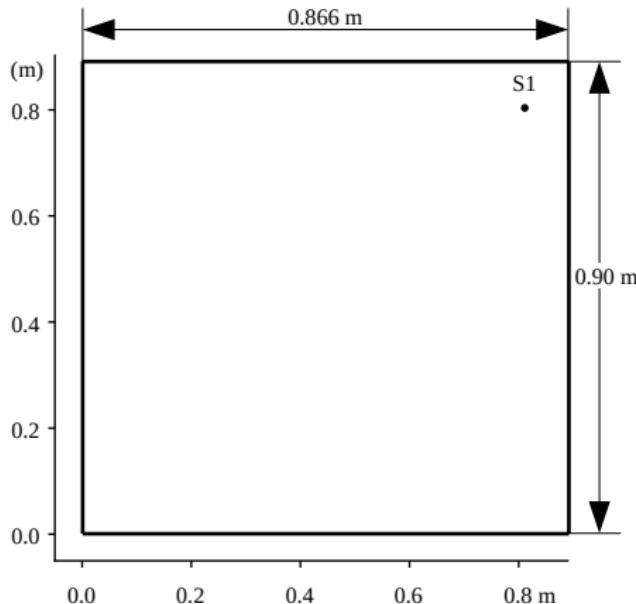
### Experimental setup:

#### Sensor and noise positions

Training dataset

Testing dataset

Model architecture



# Boosting TinyML Performance with Hybrid Floating-Point 6-bit

## TinyML Application: CNN Sensor Analytics for Structural Health Monitoring

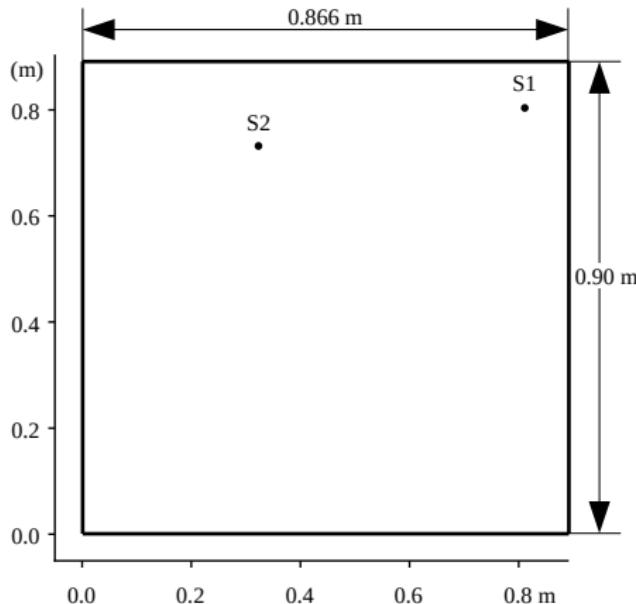
### Experimental setup:

#### Sensor and noise positions

Training dataset

Testing dataset

Model architecture



# Boosting TinyML Performance with Hybrid Floating-Point 6-bit

## TinyML Application: CNN Sensor Analytics for Structural Health Monitoring

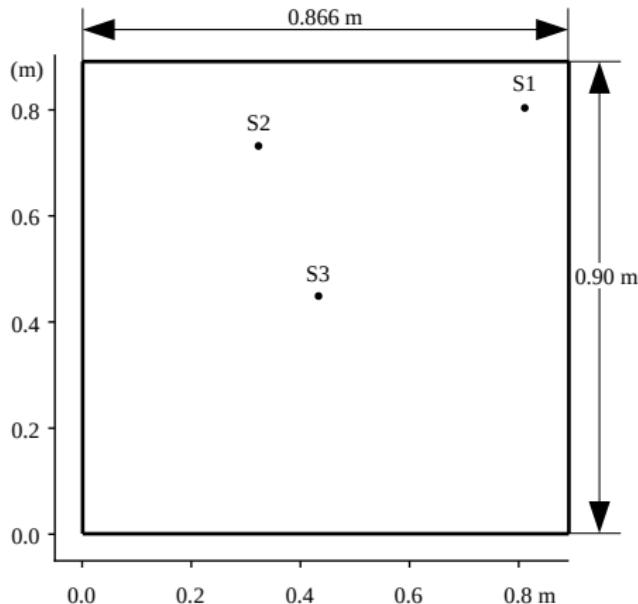
### Experimental setup:

#### Sensor and noise positions

Training dataset

Testing dataset

Model architecture



# Boosting TinyML Performance with Hybrid Floating-Point 6-bit

## TinyML Application: CNN Sensor Analytics for Structural Health Monitoring

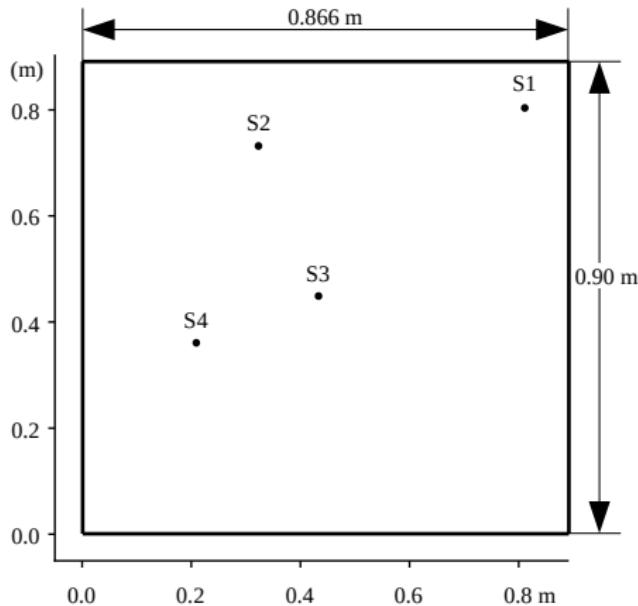
### Experimental setup:

#### Sensor and noise positions

Training dataset

Testing dataset

Model architecture



# Boosting TinyML Performance with Hybrid Floating-Point 6-bit

## TinyML Application: CNN Sensor Analytics for Structural Health Monitoring

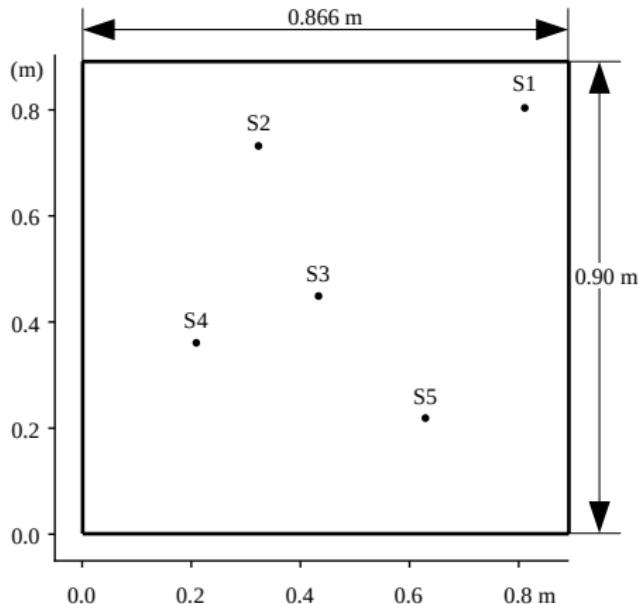
### Experimental setup:

#### Sensor and noise positions

Training dataset

Testing dataset

Model architecture



# Boosting TinyML Performance with Hybrid Floating-Point 6-bit

## TinyML Application: CNN Sensor Analytics for Structural Health Monitoring

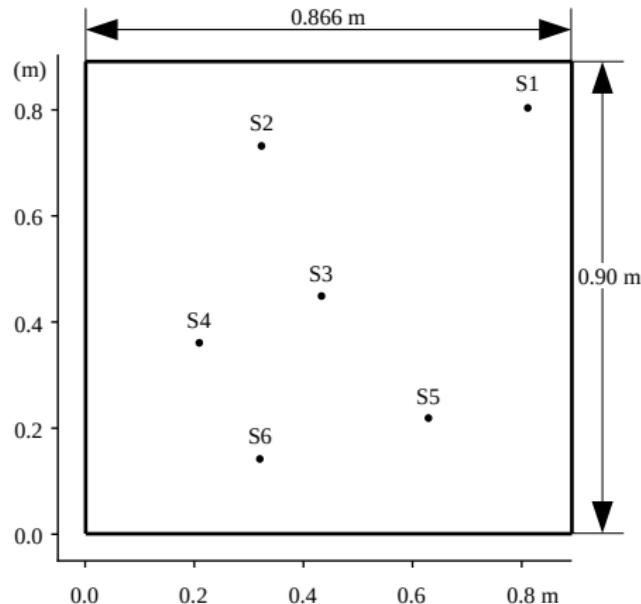
### Experimental setup:

#### Sensor and noise positions

Training dataset

Testing dataset

Model architecture



# Boosting TinyML Performance with Hybrid Floating-Point 6-bit

## TinyML Application: CNN Sensor Analytics for Structural Health Monitoring

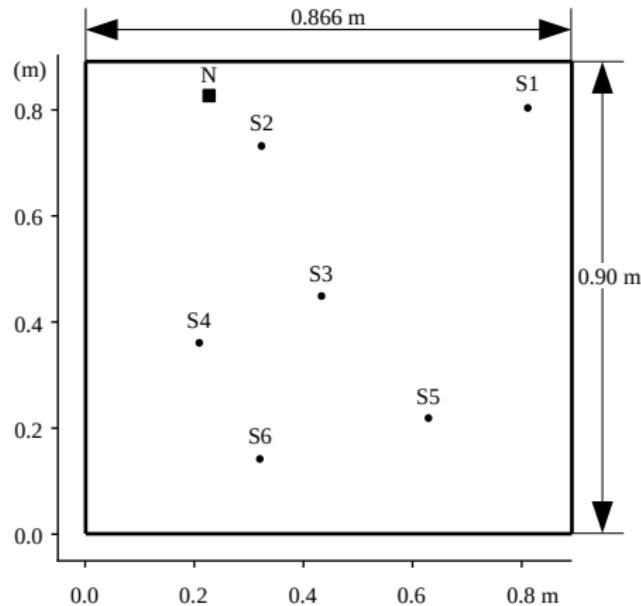
### Experimental setup:

#### Sensor and noise positions

Training dataset

Testing dataset

Model architecture



# Boosting TinyML Performance with Hybrid Floating-Point 6-bit

## TinyML Application: CNN Sensor Analytics for Structural Health Monitoring

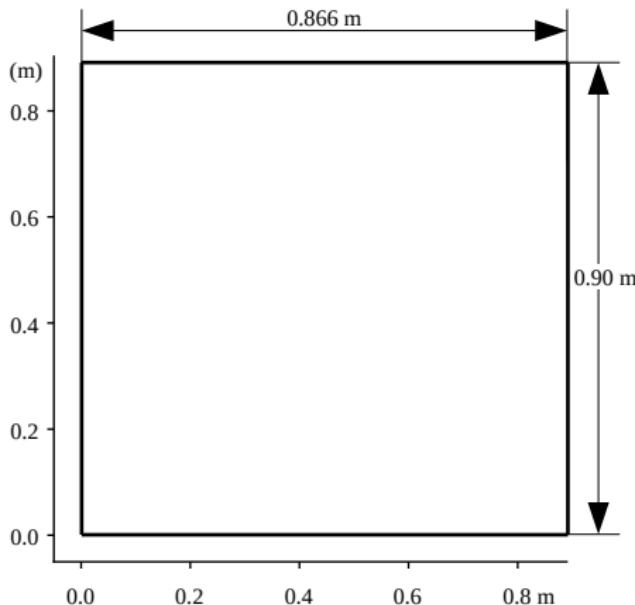
### Experimental setup:

Sensor and noise positions

**Training dataset**

Testing dataset

Model architecture



# Boosting TinyML Performance with Hybrid Floating-Point 6-bit

## TinyML Application: CNN Sensor Analytics for Structural Health Monitoring

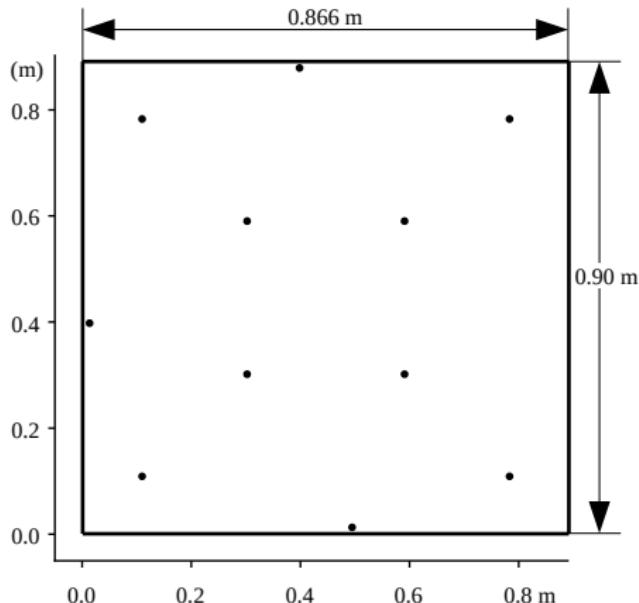
### Experimental setup:

Sensor and noise positions

**Training dataset**

Testing dataset

Model architecture



# Boosting TinyML Performance with Hybrid Floating-Point 6-bit

## TinyML Application: CNN Sensor Analytics for Structural Health Monitoring

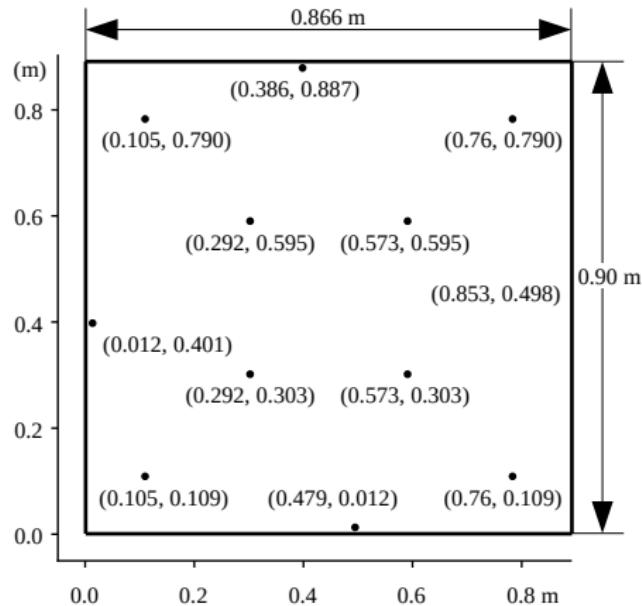
### Experimental setup:

Sensor and noise positions

**Training dataset**

Testing dataset

Model architecture



# Boosting TinyML Performance with Hybrid Floating-Point 6-bit

## TinyML Application: CNN Sensor Analytics for Structural Health Monitoring

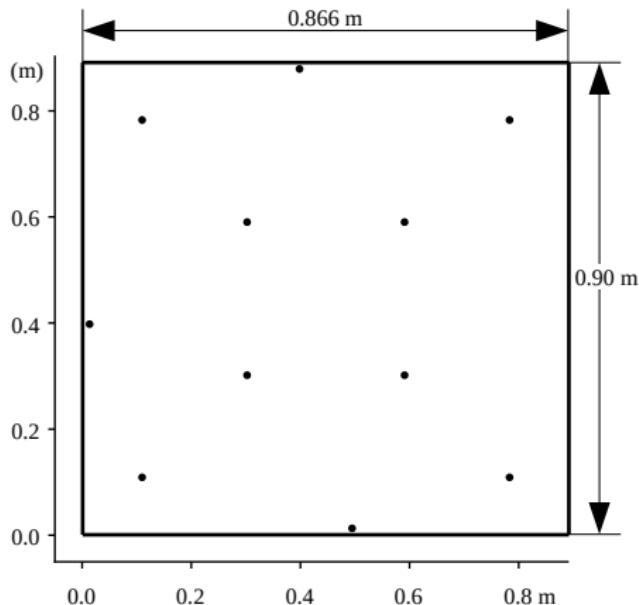
### Experimental setup:

Sensor and noise positions

**Training dataset**

Testing dataset

Model architecture



# Boosting TinyML Performance with Hybrid Floating-Point 6-bit

## TinyML Application: CNN Sensor Analytics for Structural Health Monitoring

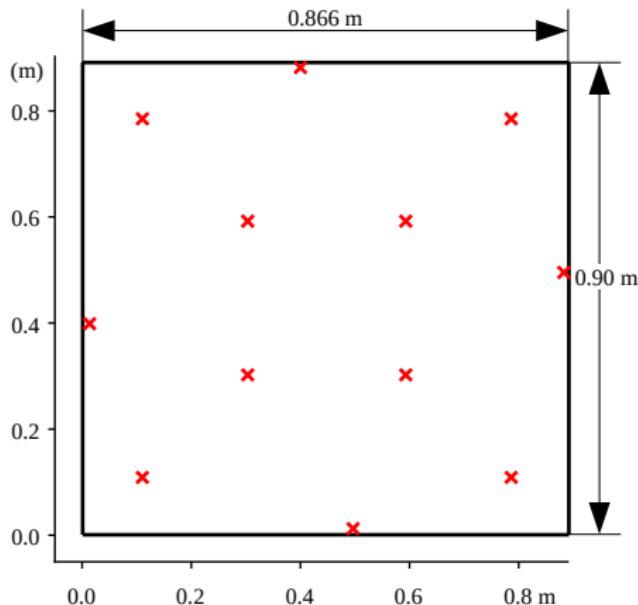
### Experimental setup:

Sensor and noise positions

Training dataset

**Testing dataset**

Model architecture



# Boosting TinyML Performance with Hybrid Floating-Point 6-bit

## TinyML Application: CNN Sensor Analytics for Structural Health Monitoring

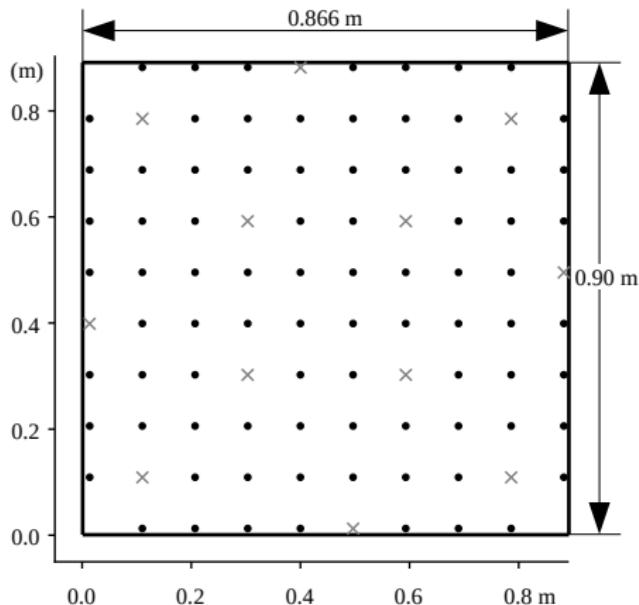
### Experimental setup:

Sensor and noise positions

Training dataset

**Testing dataset**

Model architecture



# Boosting TinyML Performance with Hybrid Floating-Point 6-bit

## TinyML Application: CNN Sensor Analytics for Structural Health Monitoring

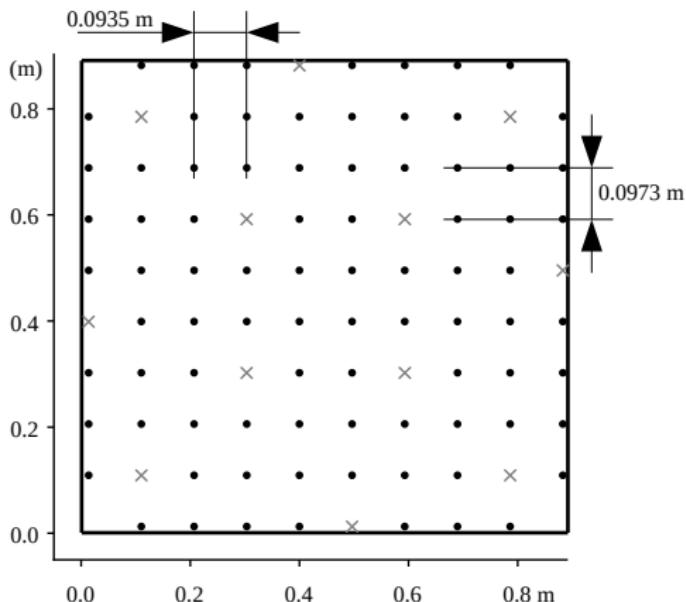
### Experimental setup:

Sensor and noise positions

Training dataset

**Testing dataset**

Model architecture



# Boosting TinyML Performance with Hybrid Floating-Point 6-bit

## TinyML Application: CNN Sensor Analytics for Structural Health Monitoring

### Experimental setup:

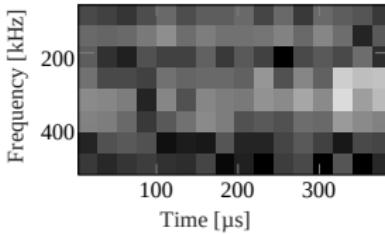
Sensor and noise positions

Training dataset

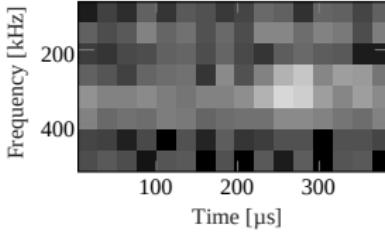
Testing dataset

**Model architecture**

S1



S2



# Boosting TinyML Performance with Hybrid Floating-Point 6-bit

## TinyML Application: CNN Sensor Analytics for Structural Health Monitoring

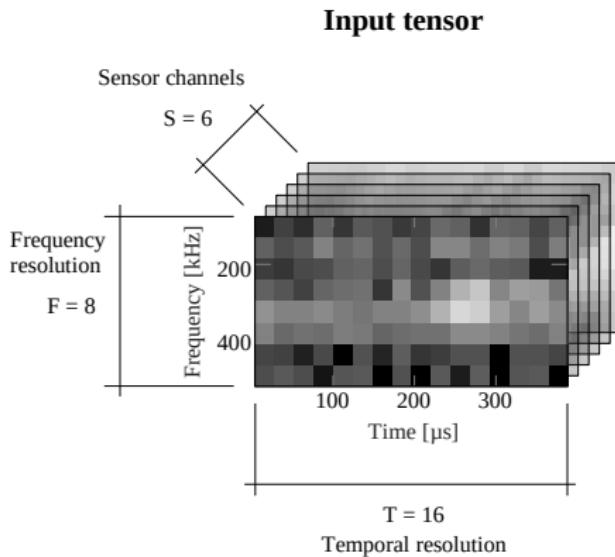
### Experimental setup:

Sensor and noise positions

Training dataset

Testing dataset

### Model architecture



# 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)

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)

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	<b>Conv<sub>c</sub></b>
2 x 2 MaxPool, stride 2	
BatchNormalization	
3 x 3 Conv (B = 55), ReLu	<b>Conv<sub>B</sub></b>
2 x 2 MaxPool, stride 2	
BatchNormalization	
3 x 3 Conv (A = 50), ReLu	<b>Conv<sub>A</sub></b>
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

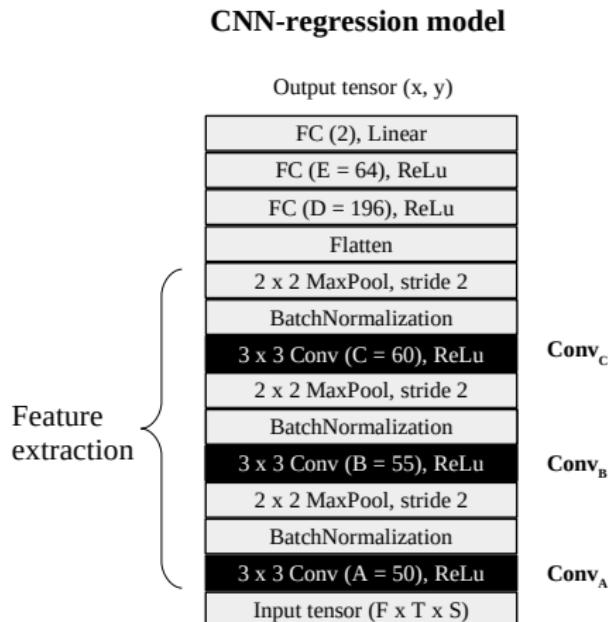
### Experimental setup:

Sensor and noise positions

Training dataset

Testing dataset

### Model architecture



# Boosting TinyML Performance with Hybrid Floating-Point 6-bit

## TinyML Application: CNN Sensor Analytics for Structural Health Monitoring

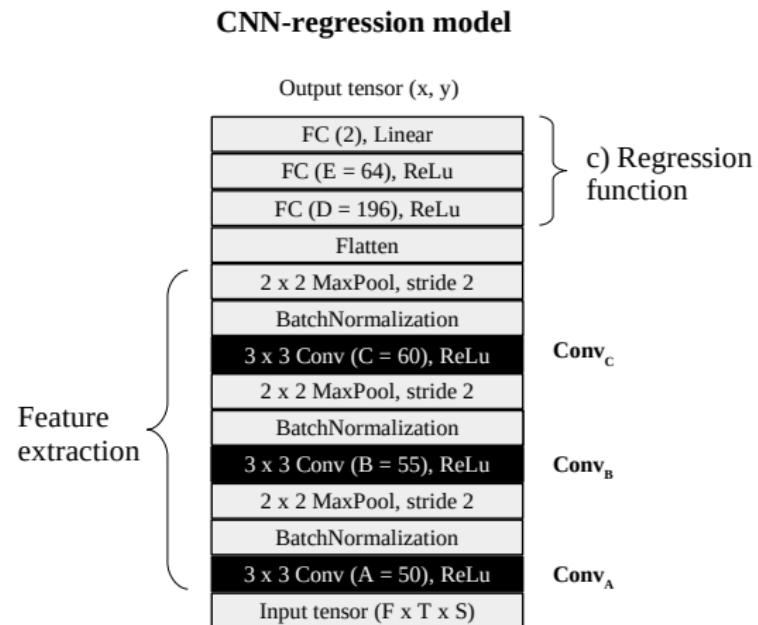
### Experimental setup:

Sensor and noise positions

Training dataset

Testing dataset

### Model architecture



# Boosting TinyML Performance with Hybrid Floating-Point 6-bit

## TinyML Application: CNN Sensor Analytics for Structural Health Monitoring

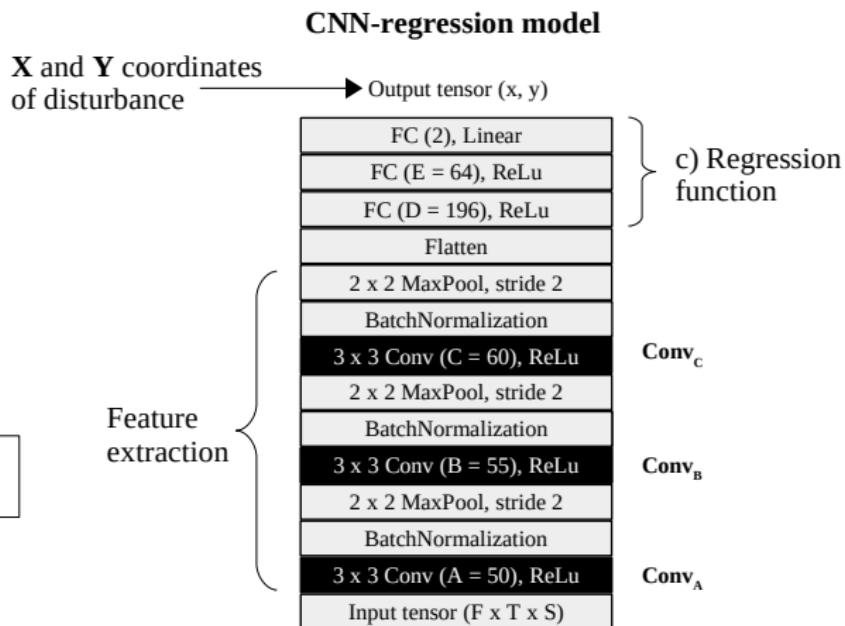
### Experimental setup:

Sensor and noise positions

Training dataset

Testing dataset

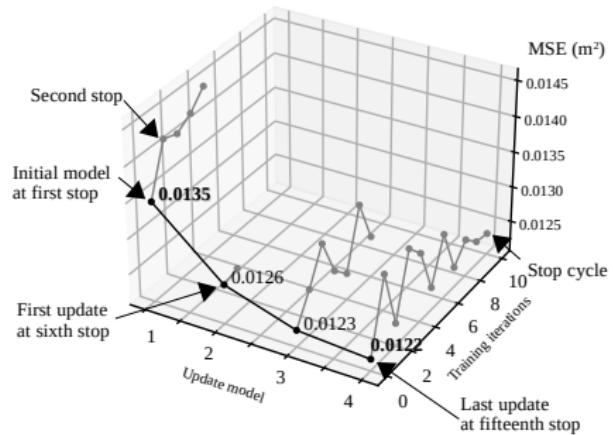
### Model architecture



# Boosting TinyML Performance with Hybrid Floating-Point 6-bit

## Multi-Phase Model Optimization

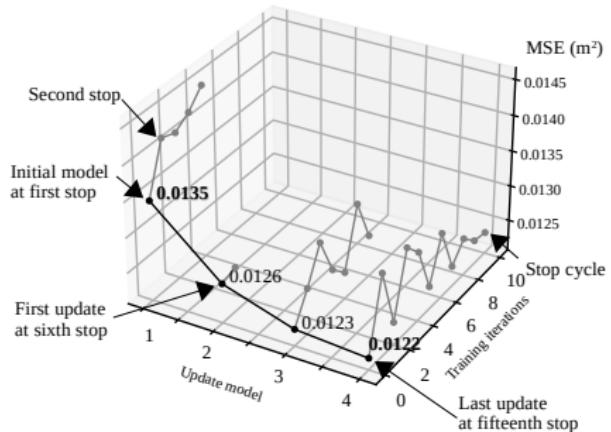
Training with iterative early stop  
with Adam resets



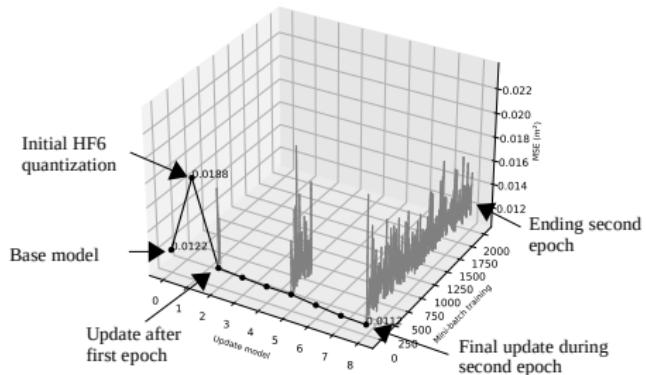
# Boosting TinyML Performance with Hybrid Floating-Point 6-bit

## Multi-Phase Model Optimization

Training with iterative early stop  
with Adam resets

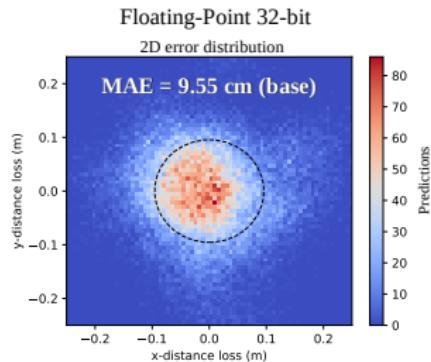


Quantization-aware training for  
hybrid floating-point 6-bit



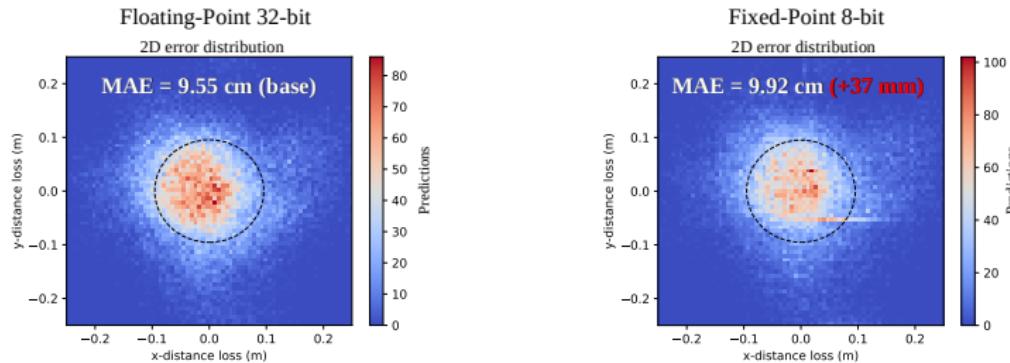
# Boosting TinyML Performance with Hybrid Floating-Point 6-bit

## Assessing X and Y Coordinate Prediction Accuracy Across Multiple Quantization Strategies



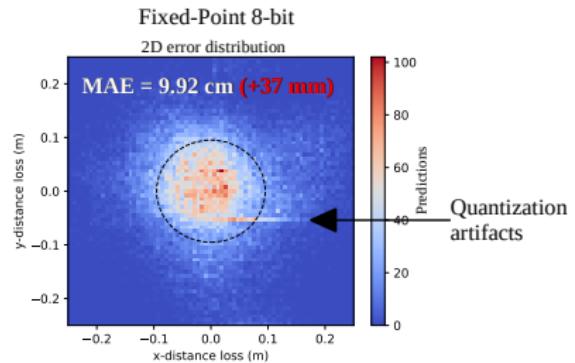
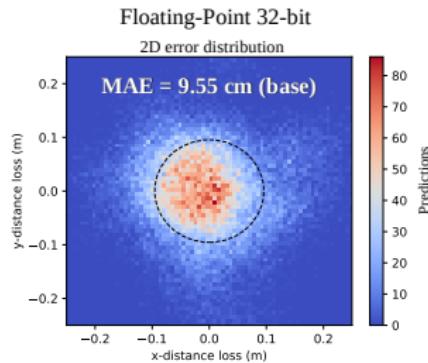
# Boosting TinyML Performance with Hybrid Floating-Point 6-bit

## Assessing X and Y Coordinate Prediction Accuracy Across Multiple Quantization Strategies



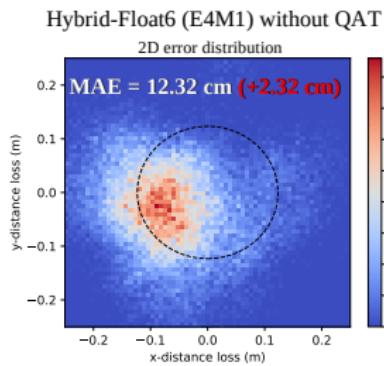
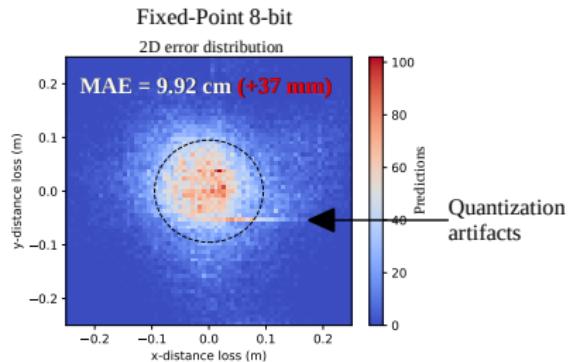
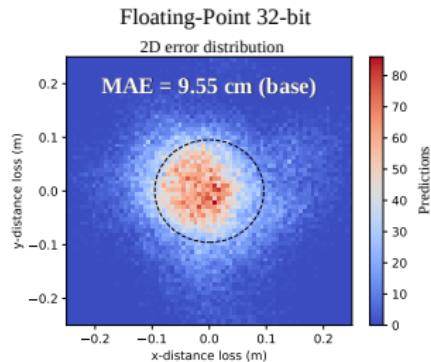
# Boosting TinyML Performance with Hybrid Floating-Point 6-bit

## Assessing X and Y Coordinate Prediction Accuracy Across Multiple Quantization Strategies



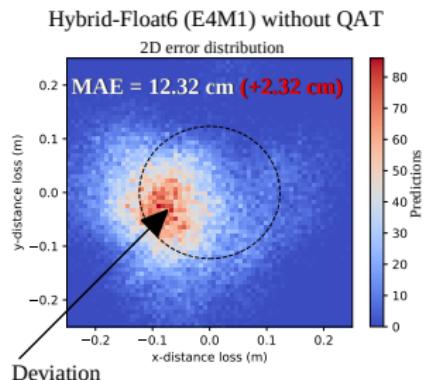
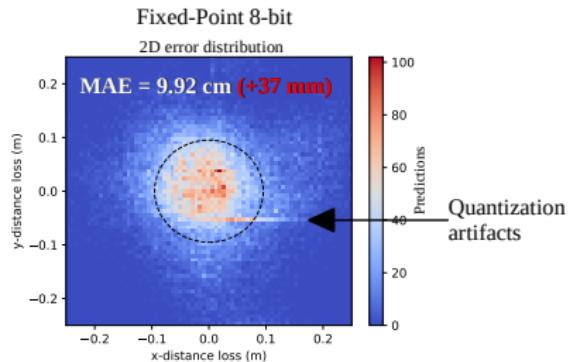
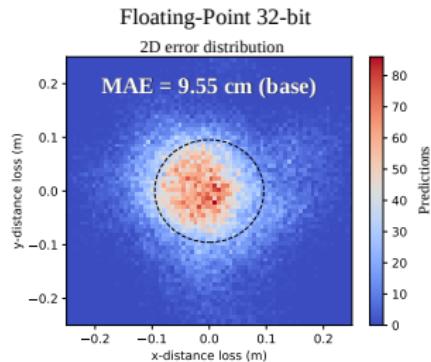
# Boosting TinyML Performance with Hybrid Floating-Point 6-bit

## Assessing X and Y Coordinate Prediction Accuracy Across Multiple Quantization Strategies



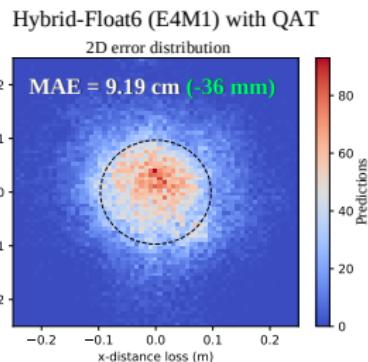
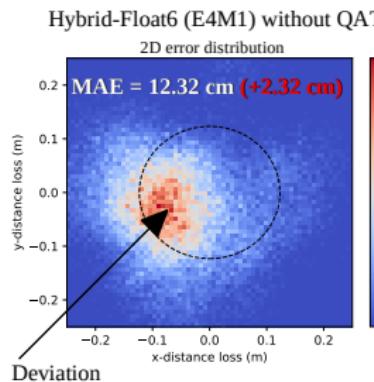
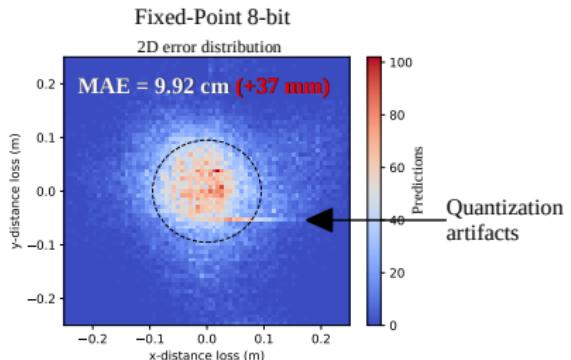
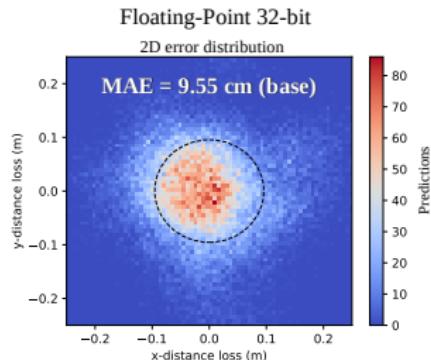
# Boosting TinyML Performance with Hybrid Floating-Point 6-bit

## Assessing X and Y Coordinate Prediction Accuracy Across Multiple Quantization Strategies



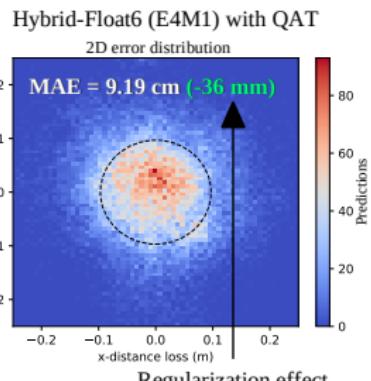
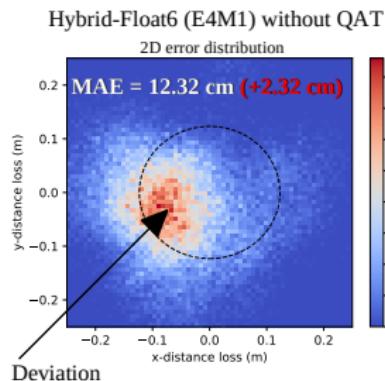
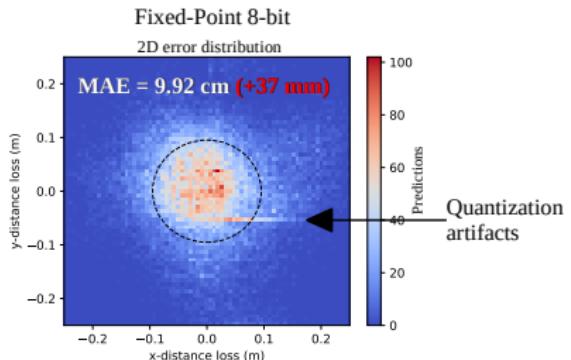
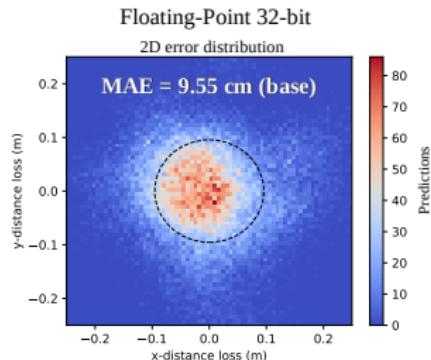
# Boosting TinyML Performance with Hybrid Floating-Point 6-bit

## Assessing X and Y Coordinate Prediction Accuracy Across Multiple Quantization Strategies



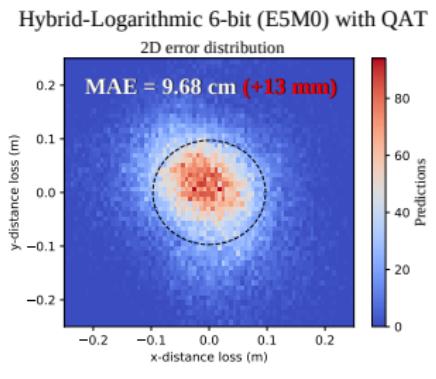
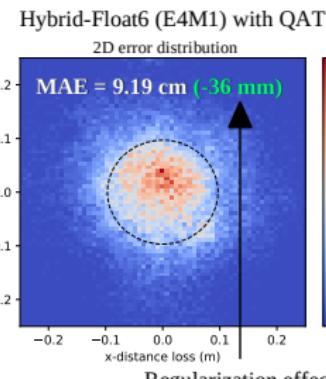
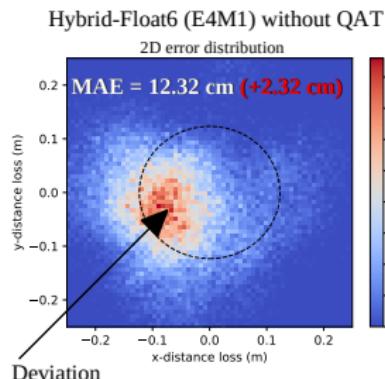
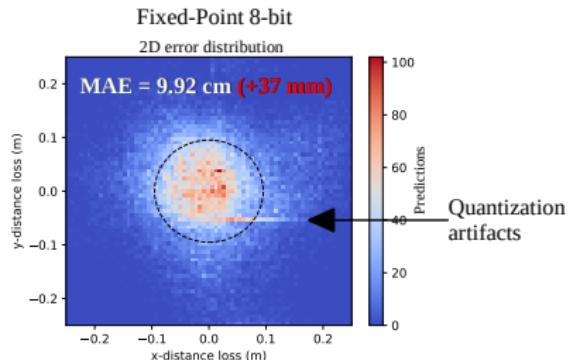
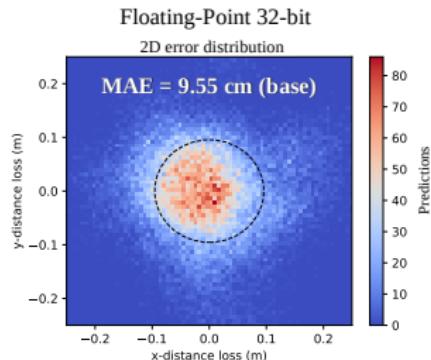
# Boosting TinyML Performance with Hybrid Floating-Point 6-bit

## Assessing X and Y Coordinate Prediction Accuracy Across Multiple Quantization Strategies



# Boosting TinyML Performance with Hybrid Floating-Point 6-bit

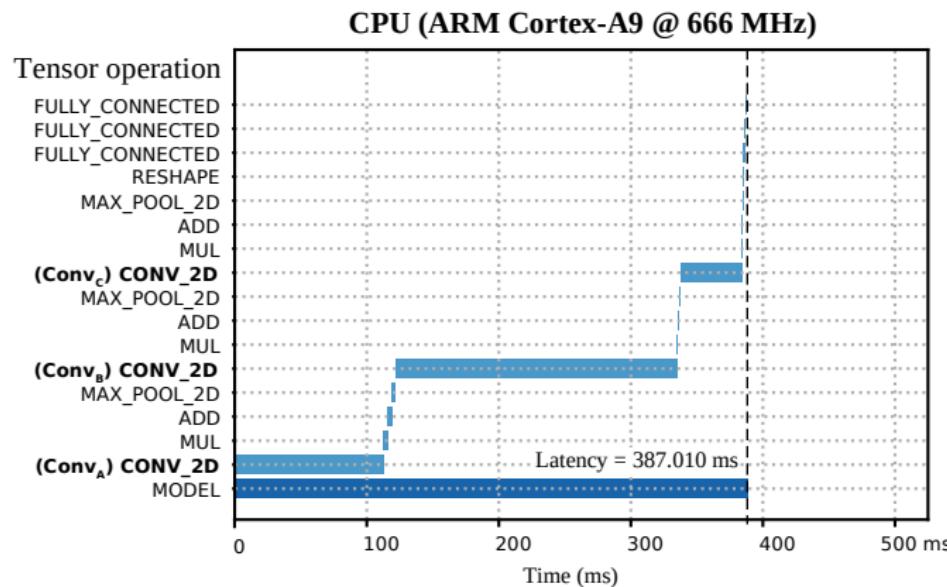
## Assessing X and Y Coordinate Prediction Accuracy Across Multiple Quantization Strategies



# Boosting TinyML Performance with Hybrid Floating-Point 6-bit

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

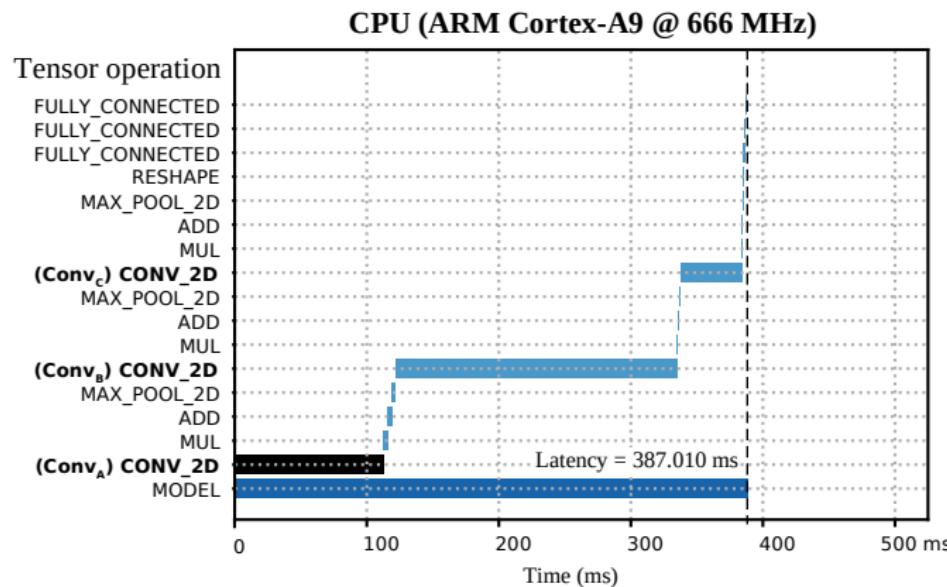
Tensor Op	CPU (ms)	TP FP32 (ms)	Gain	TP HF6 (ms)	Gain
Conv <sub>A</sub>					
Conv <sub>B</sub>					
Conv <sub>C</sub>					



# Boosting TinyML Performance with Hybrid Floating-Point 6-bit

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

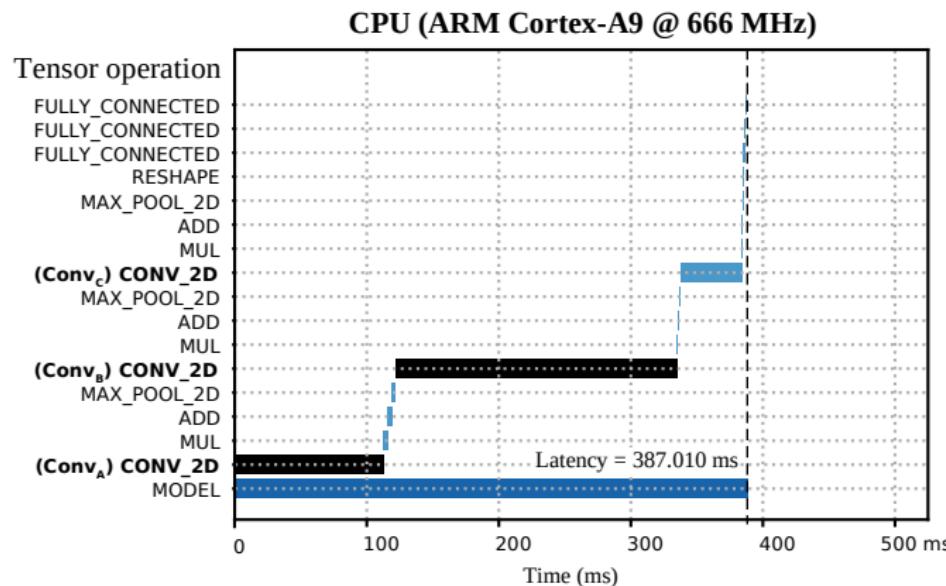
Tensor Op	CPU (ms)	TP FP32 (ms)	Gain	TP HF6 (ms)	Gain
Conv <sub>A</sub>	112.24				
Conv <sub>B</sub>					
Conv <sub>C</sub>					



# Boosting TinyML Performance with Hybrid Floating-Point 6-bit

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

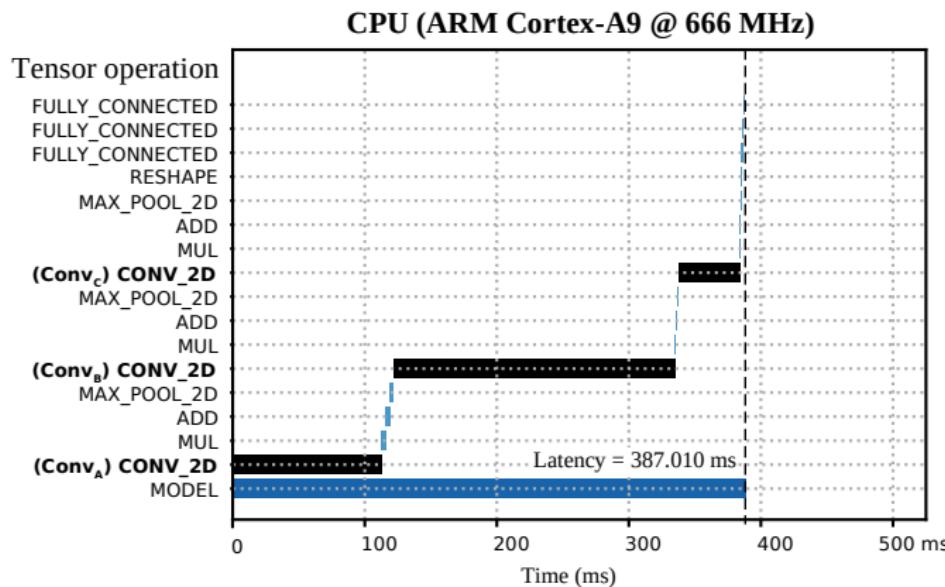
Tensor Op	CPU (ms)	TP FP32 (ms)	Gain	TP HF6 (ms)	Gain
Conv <sub>A</sub>	112.24				
Conv <sub>B</sub>	213.13				
Conv <sub>C</sub>					



# Boosting TinyML Performance with Hybrid Floating-Point 6-bit

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

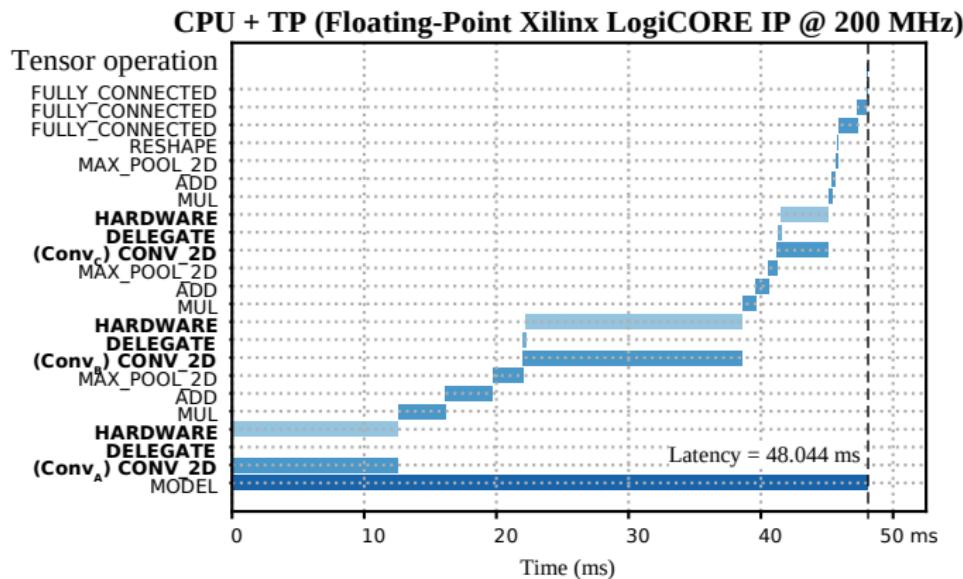
Tensor Op	CPU (ms)	TP FP32 (ms)	Gain	TP HF6 (ms)	Gain
Conv <sub>A</sub>	112.24				
Conv <sub>B</sub>	213.13				
Conv <sub>C</sub>	46.59				



# Boosting TinyML Performance with Hybrid Floating-Point 6-bit

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

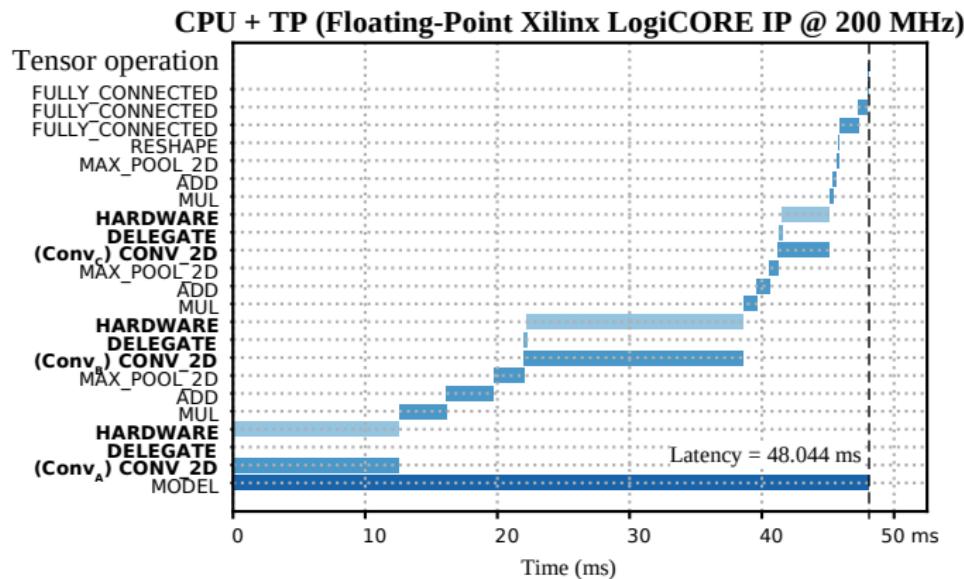
Tensor Op	CPU (ms)	TP FP32 (ms)	Gain	TP HF6 (ms)	Gain
Conv <sub>A</sub>	112.24				
Conv <sub>B</sub>	213.13				
Conv <sub>C</sub>	46.59				



# Boosting TinyML Performance with Hybrid Floating-Point 6-bit

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

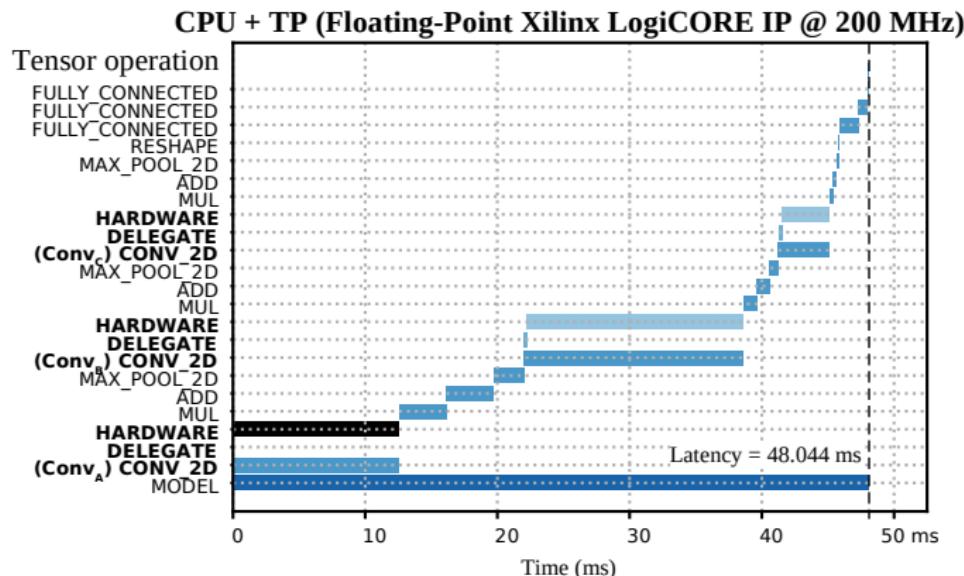
Tensor Op	CPU (ms)	TP FP32 (ms)	Gain	TP HF6 (ms)	Gain
Conv <sub>A</sub>	112.24				
Conv <sub>B</sub>	213.13				
Conv <sub>C</sub>	46.59				



# Boosting TinyML Performance with Hybrid Floating-Point 6-bit

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

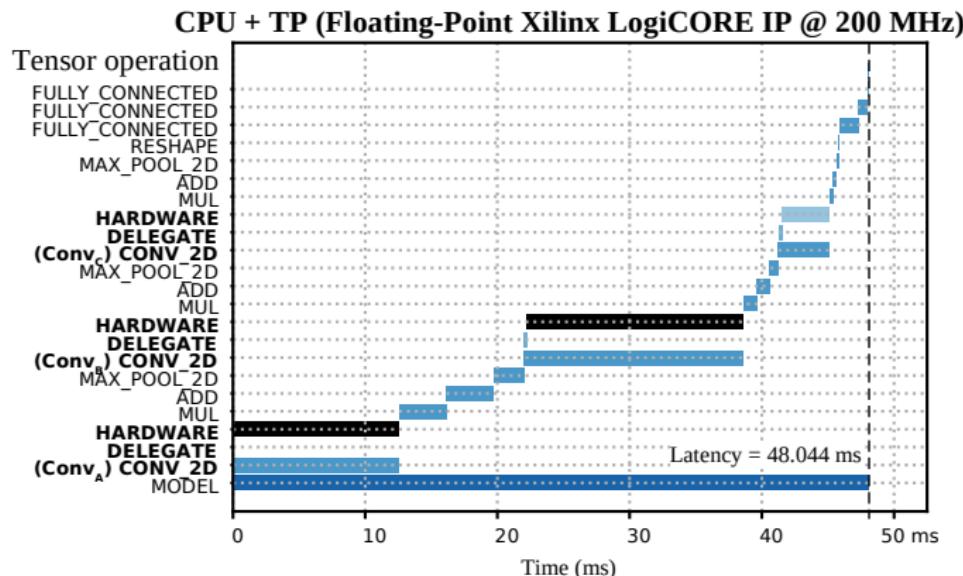
Tensor Op	CPU (ms)	TP FP32 (ms)	Gain	TP HF6 (ms)	Gain
Conv <sub>A</sub>	112.24	12.49	8.9		
Conv <sub>B</sub>	213.13				
Conv <sub>C</sub>	46.59				



# Boosting TinyML Performance with Hybrid Floating-Point 6-bit

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

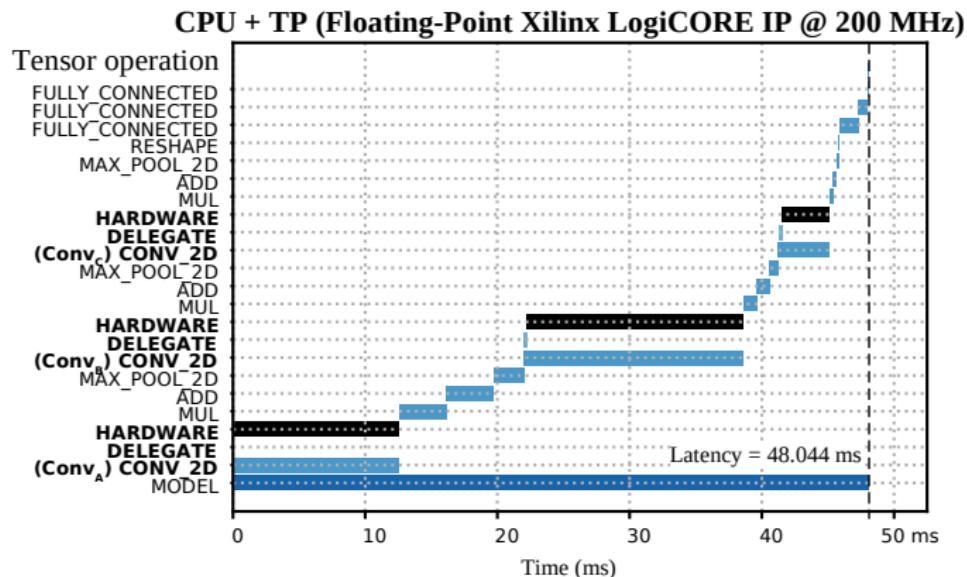
Tensor Op	CPU (ms)	TP FP32 (ms)	Gain	TP HF6 (ms)	Gain
Conv <sub>A</sub>	112.24	12.49	8.9		
Conv <sub>B</sub>	213.13	16.39	13.0		
Conv <sub>C</sub>	46.59				



# Boosting TinyML Performance with Hybrid Floating-Point 6-bit

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

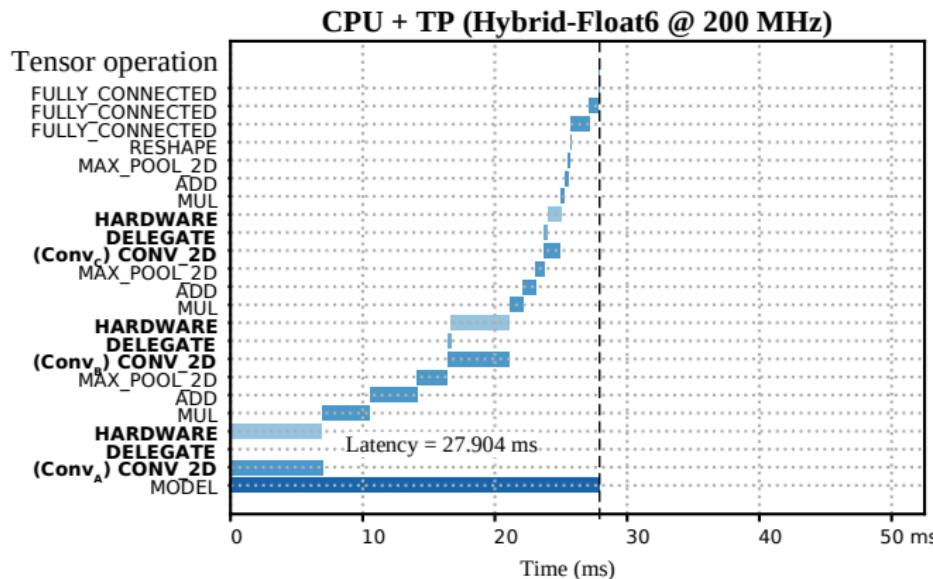
Tensor Op	CPU (ms)	TP FP32 (ms)	Gain	TP HF6 (ms)	Gain
Conv <sub>A</sub>	112.24	12.49	8.9		
Conv <sub>B</sub>	213.13	16.39	13.0		
Conv <sub>C</sub>	46.59	3.59	12.9		



# Boosting TinyML Performance with Hybrid Floating-Point 6-bit

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

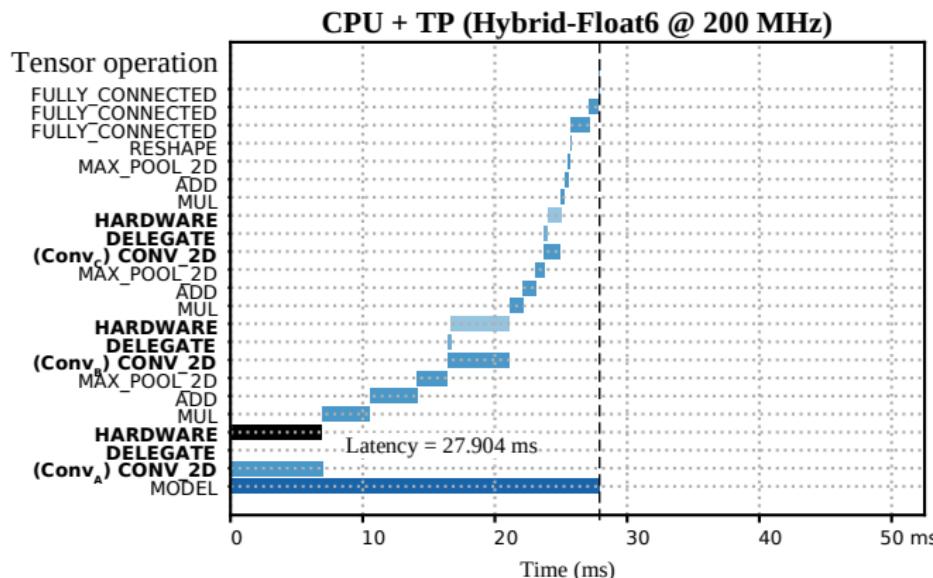
Tensor Op	CPU (ms)	TP FP32 (ms)	Gain	TP HF6 (ms)	Gain
Conv <sub>A</sub>	112.24	12.49	8.9		
Conv <sub>B</sub>	213.13	16.39	13.0		
Conv <sub>C</sub>	46.59	3.59	12.9		



# Boosting TinyML Performance with Hybrid Floating-Point 6-bit

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

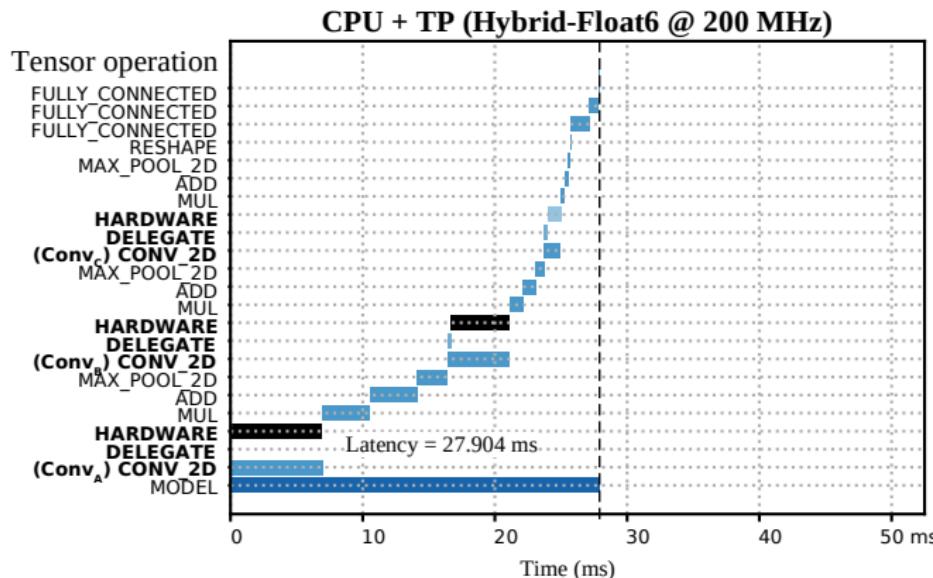
Tensor Op	CPU (ms)	TP FP32 (ms)	Gain	TP HF6 (ms)	Gain
Conv <sub>A</sub>	112.24	12.49	8.9	<b>6.92</b>	<b>16.2</b>
Conv <sub>B</sub>	213.13	16.39	13.0		
Conv <sub>C</sub>	46.59	3.59	12.9		



# Boosting TinyML Performance with Hybrid Floating-Point 6-bit

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

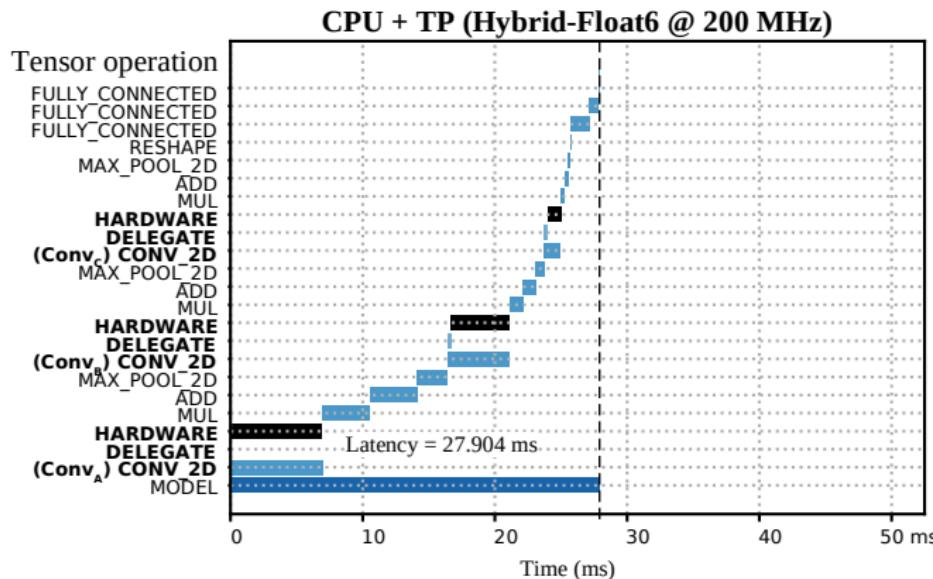
Tensor Op	CPU (ms)	TP FP32 (ms)	Gain	TP HF6 (ms)	Gain
Conv <sub>A</sub>	112.24	12.49	8.9	<b>6.92</b>	<b>16.2</b>
Conv <sub>B</sub>	213.13	16.39	13.0	<b>4.41</b>	<b>48.3</b>
Conv <sub>C</sub>	46.59	3.59	12.9		



# Boosting TinyML Performance with Hybrid Floating-Point 6-bit

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

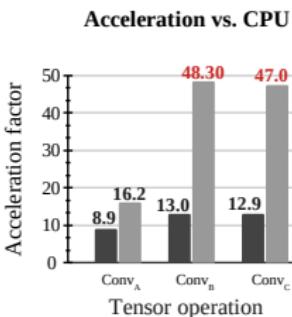
Tensor Op	CPU (ms)	TP FP32 (ms)	Gain	TP HF6 (ms)	Gain
Conv <sub>A</sub>	112.24	12.49	8.9	<b>6.92</b>	<b>16.2</b>
Conv <sub>B</sub>	213.13	16.39	13.0	<b>4.41</b>	<b>48.3</b>
Conv <sub>C</sub>	46.59	3.59	12.9	<b>0.99</b>	<b>47.0</b>



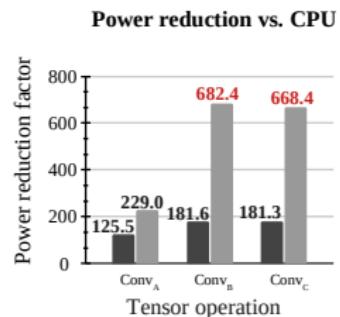
# Boosting TinyML Performance with Hybrid Floating-Point 6-bit

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

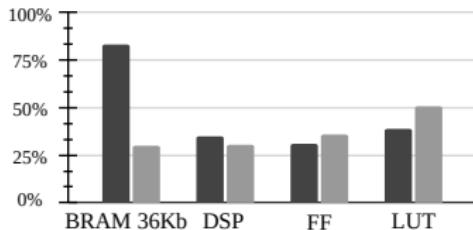
■ Floating-Point  
■ Hybrid-Float6



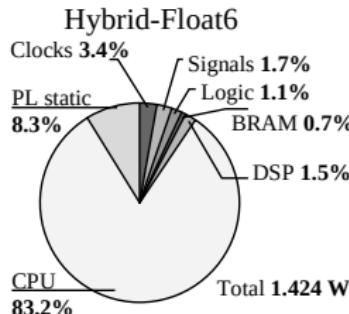
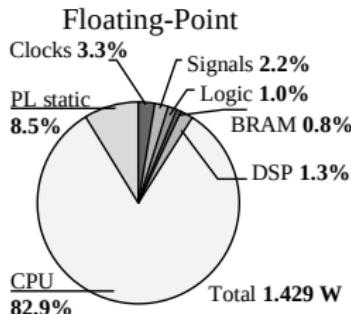
Smallest CNN floating-point accelerator in the literature



Hardware resource utilization.



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

## Conclusion and Future Research

- **HW/SW Design Methodology for Low-Power Neural Network Acceleration with Hybrid Custom Floating-Point Arithmetic**
- **Technological Transformation**

## Conclusion and Future Research

- **HW/SW Design Methodology for Low-Power Neural Network Acceleration with Hybrid Custom Floating-Point Arithmetic**
  - Mimic standard floating-point acceleration in extreme edge devices
- **Technological Transformation**

## Conclusion and Future Research

- **HW/SW Design Methodology for Low-Power Neural Network Acceleration with Hybrid Custom Floating-Point Arithmetic**
  - Mimic standard floating-point acceleration in extreme edge devices
  - Smallest CNN floating-point accelerator in the literature
- **Technological Transformation**

## Conclusion and Future Research

- **HW/SW Design Methodology for Low-Power Neural Network Acceleration with Hybrid Custom Floating-Point Arithmetic**
  - Mimic standard floating-point acceleration in extreme edge devices
  - Smallest CNN floating-point accelerator in the literature
  - On-device iterative optimization with non-negativity constraints using 4-bit weights
- **Technological Transformation**

## Conclusion and Future Research

- **HW/SW Design Methodology for Low-Power Neural Network Acceleration with Hybrid Custom Floating-Point Arithmetic**

- Mimic standard floating-point acceleration in extreme edge devices
- Smallest CNN floating-point accelerator in the literature
- On-device iterative optimization with non-negativity constraints using 4-bit weights

- **Technological Transformation**

- Inference was the beginning in TinyML. Low-precision floating-point is opening the door for a new generation of on-device training accelerators that are essential in the next phase of AI evolution.

# 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](https://doi.org/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](https://doi.org/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](https://doi.org/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](https://doi.org/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