Sound Mixed Fixed-Point Quantization of Neural Networks

Debasmita Lohar, Clothilde Jeangoudoux, Anastasia Volkova, Eva Darulova

EMSOFT 2023
Neural networks are ubiquitous in safety-critical systems!

- Adaptive Cruise Control
- Collision Avoidance System
- Translational Oscillator Controller
- Drone Controller
Neural Networks as Controllers

closed-loop system

plant model

sensors

actuators

controller
Neural Networks as Controllers

$$x_1 = \text{relu}(W_1 \times [\text{in}] + [b_1])$$

$$\text{out} = \text{linear}(W_2 \times [x_1] + [b_2])$$

feed-forward regression model
Neural networks are usually trained in high-precision!
Neural networks are usually trained in high-precision!
Model is usually in high-precision!

\[ x_1 = \text{relu}(W_1 \times \text{in} + b_1) \]
\[ \text{out} = \text{linear}(W_2 \times x_1 + b_2) \]
Model is usually in high-precision!

\[
x_1 = \text{relu}(W_1 x + b_1)
\]

\[
\text{out} = \text{linear}(W_2 x_1 + b_2)
\]

64-bit floating-point
Safety Verification of Neural Network Controllers

- ReachNN ’19
- Sherlock ’19
- NNV ’20
- ...

input data → training

high-precision GPU
Safety Verification of Neural Network Controllers

- ReachNN '19
- Sherlock '19
- NNV '20
- ...

real-valued arithmetic
Safety Verification of Neural Network Controllers

Do not directly consider the finite-precision deployment of the model.
Controllers deployed in Embedded Systems

- High-precision model
- Safety +/- error
- Fixed low precision system

Input data → Training → High-precision model → Model deployment
Controllers deployed in Embedded Systems

- High-precision GPU
- Input data 
- Training
- High-precision model
- Quantization
- Safety +/- error
- Fixed low precision system
- Model deployment
Neural Network Quantization

Our Goal: Quantize respecting the error bound!
Example: Unicycle Controller

- $-0.6 \leq in1 \leq 9.55$
- $-4.5 \leq in2 \leq 0.2$
- $-0.06 \leq in3 \leq 2.11$
- $-0.3 \leq in4 \leq 1.51$

res $\pm 1e^{-3}$

500 neurons
Example: Unicycle Controller

\[-0.6 \leq \text{in1} \leq 9.55\]
\[-4.5 \leq \text{in2} \leq 0.2\]
\[-0.06 \leq \text{in3} \leq 2.11\]
\[-0.3 \leq \text{in4} \leq 1.51\]

```
#include <math.h>
#include <ap_fixed.h>
#include <hls_math.h>
#include <ap_fixed.h>

void nn1(ap_fixed<24,5> x_0, ap_fixed<24,4> x_1, ap_fixed<24,3> x_2,
ap_fixed<24,2> x_3, ap_fixed<27,8> _result[2]) {
    ap_fixed<24,1> weights1_0_0 = -0.036691424;
    ...
    ap_fixed<27,8> layer2_dot_1 = (_tmp4994 + _tmp4995);
    ap_fixed<27,8> layer2_bias_0 = (layer2_dot_0 + (ap_fixed<27,1>) (bias2_0));
    ap_fixed<27,8> layer2_bias_1 = (layer2_dot_1 + (ap_fixed<27,1>) (bias2_1));
    ap_fixed<27,8> layer2_0 = (layer2_bias_0);
    ap_fixed<27,8> layer2_1 = (layer2_bias_1);
    _result[0] = layer2_0;
    _result[1] = layer2_1;
}
```
Example: Unicycle Controller

-0.6 <= in1 <= 9.55
-4.5 <= in2 <= 0.2
-0.06 <= in3 <= 2.11
-0.3 <= in4 <= 1.51

mixed-precision fixed-point code

```c
#include <math.h>
#include <ap_fixed.h>
#include <hls_math.h>

void nn1(ap_fixed<24,5> x_0, ap_fixed<24,4> x_1, ap_fixed<24,3> x_2,
         ap_fixed<24,2> x_3, ap_fixed<27,8> _result[2]) {
    ap_fixed<24,1> weights1_0_0 = -0.036691424;
    ...
    ap_fixed<27,8> layer2_dot_1 = (_tmp4994 + _tmp4995);
    ap_fixed<27,8> layer2_bias_0 = (layer2_dot_0 + (ap_fixed<27,1> (bias2_0)));
    ap_fixed<27,8> layer2_bias_1 = (layer2_dot_1 + (ap_fixed<27,1> (bias2_1)));
    ap_fixed<27,8> layer2_0 = (layer2_bias_0);
    ap_fixed<27,8> layer2_1 = (layer2_bias_1);
    _result[0] = layer2_0;
    _result[1] = layer2_1;
}
```
Example: Unicycle Controller

-0.6 <= in1 <= 9.55
-4.5 <= in2 <= 0.2
-0.06 <= in3 <= 2.11
-0.3 <= in4 <= 1.51

mixed-precision fixed-point code

```c
#include <math.h>
#include <ap_fixed.h>
#include <hls_math.h>
#include <ap_fixed.h>

void nn1(ap_fixed<24,5> x_0, ap_fixed<24,4> x_1, ap_fixed<24,3> x_2,
         ap_fixed<24,2> x_3, ap_fixed<27,8> _result[2]) {
    ap_fixed<24,1> weights1_0_0 = -0.036691424;
    ...

    ap_fixed<27,8> layer2_dot_1 = (_tmp4994 + _tmp4995);
    ap_fixed<27,8> layer2_bias_0 = (layer2_dot_0 + (ap_fixed<27,8>) (bias2_0));
    ap_fixed<27,8> layer2_bias_1 = (layer2_dot_1 + (ap_fixed<27,8>) (bias2_1));
    ap_fixed<27,8> layer2_0 = (layer2_bias_0);
    ap_fixed<27,8> layer2_1 = (layer2_bias_1);
    _result[0] = layer2_0;
    _result[1] = layer2_1;
}
```

directly compiled

latency = 27 cycles
State-of-the-Art in Sound Mixed Precision Tuning

-0.6 \leq in1 \leq 9.55
-4.5 \leq in2 \leq 0.2
-0.06 \leq in3 \leq 2.11
-0.3 \leq in4 \leq 1.51

res +/− 1e−3

latency = 178 cycles

1. Daisy - Framework for Analysis and Optimization of Numerical Programs, E. Darulova et al., TACAS 2018
State-of-the-Art in Sound Mixed Precision Tuning

\[-0.6 \leq \text{in1} \leq 9.55\]
\[-4.5 \leq \text{in2} \leq 0.2\]
\[-0.06 \leq \text{in3} \leq 2.11\]
\[-0.3 \leq \text{in4} \leq 1.51\]

\[\text{res } +/−\ 1e-3\]

needs unrolled structures

Daisy

latency = 178 cycles

over-approximates a lot!

1. Daisy - Framework for Analysis and Optimization of Numerical Programs, E. Darulova et al., TACAS 2018
State-of-the-Art in Sound Mixed Precision Tuning

\[-0.6 \leq \text{in1} \leq 9.55\]
\[-4.5 \leq \text{in2} \leq 0.2\]
\[-0.06 \leq \text{in3} \leq 2.11\]
\[-0.3 \leq \text{in4} \leq 1.51\]

1. Daisy - Framework for Analysis and Optimization of Numerical Programs, E. Darulova et al., TACAS 2018
2. Rigorous floating-point mixed-precision tuning, W. Chiang et al., POPL 2017
-0.6 <= in1 <= 9.55
-4.5 <= in2 <= 0.2
-0.06 <= in3 <= 2.11
-0.3 <= in4 <= 1.51

State-of-the-Art in Sound Mixed Precision Tuning

1. Daisy - Framework for Analysis and Optimization of Numerical Programs, E. Darulova et al., TACAS 2018
2. Rigorous floating-point mixed-precision tuning, W. Chiang et al., POPL 2017

res +/- 1e-3

no fixed-point support!
State-of-the-Art in Neural Network Quantization

-0.6 <= in1 <= 9.55
-4.5 <= in2 <= 0.2
-0.06 <= in3 <= 2.11
-0.3 <= in4 <= 1.51

1. Compiling KB-Sized Machine Learning Models to Tiny IoT Devices, S. Gopinath et al., PLDI 2019
2. Shiftry: RNN inference in 2KB of RAM, A. Kumar et al., OOPSLA 2020
State-of-the-Art in Neural Network Quantization

-0.6 <= in1 <= 9.55
-4.5 <= in2 <= 0.2
-0.06 <= in3 <= 2.11
-0.3 <= in4 <= 1.51

res +/- 1e-3

- Shiftry$^2$
- SeeDot$^1$

FPTuner$^2$
Daisy$^1$

does not work for controllers!

1. Compiling KB-Sized Machine Learning Models to Tiny IoT Devices, S. Gopinath et al., PLDI 2019
2. Shiftry: RNN inference in 2KB of RAM, A. Kumar et al., OOPSLA 2020
State-of-the-Art in Neural Network Quantization

-0.6 \leq in1 \leq 9.55
-4.5 \leq in2 \leq 0.2
-0.06 \leq in3 \leq 2.11
-0.3 \leq in4 \leq 1.51

res +/− 1e-3

Daisy

Daisy1

FPTuner

FPTuner2

SeeDot1

Shiftry2

... unsound! does not work for controllers!

1. Compiling KB-Sized Machine Learning Models to Tiny IoT Devices, S. Gopinath et al., PLDI 2019
2. Shiftry: RNN inference in 2KB of RAM, A. Kumar et al., OOPSLA 2020
State-of-the-Art is not enough!

\[-0.6 \leq \text{in1} \leq 9.55\]
\[-4.5 \leq \text{in2} \leq 0.2\]
\[-0.06 \leq \text{in3} \leq 2.11\]
\[-0.3 \leq \text{in4} \leq 1.51\]

\[\text{res} \pm \text{1e-3}\]

sound tuner for numerical programs

unsound quantizers for classifiers

Daisy

FPTuner

SeeDot

Shiftry
State-of-the-Art is not enough!

-0.6 <= in1 <= 9.55
-4.5 <= in2 <= 0.2
-0.06 <= in3 <= 2.11
-0.3 <= in4 <= 1.51

res +/- 1e-3

- We provide: sound quantizer for NN controllers guaranteeing error bounds
Key Idea: Quantization for efficiency is an optimization problem!

\[-0.6 \leq \text{in1} \leq 9.55\]
\[-4.5 \leq \text{in2} \leq 0.2\]
\[-0.06 \leq \text{in3} \leq 2.11\]
\[-0.3 \leq \text{in4} \leq 1.51\]

\[
\text{res} +/- 1e-3
\]

\[
\text{minimize: precision}
\]
Key Idea: Quantization for efficiency is an optimization problem!

\[-0.6 \leq \text{in1} \leq 9.55\]
\[-4.5 \leq \text{in2} \leq 0.2\]
\[-0.06 \leq \text{in3} \leq 2.11\]
\[-0.3 \leq \text{in4} \leq 1.51\]

\[
\minimize \gamma = \sum_{i=1}^{n} \gamma_i^{\text{dot}} + \gamma_i^{\text{bias}} + \gamma_i^{\alpha}
\]

- integer-valued cost

res +/- 1e-3
Key Idea: Quantization for efficiency is an optimization problem!

\[ \min \gamma = \sum_{i=1}^{n} \gamma_{i}^{\text{dot}} + \gamma_{i}^{\text{bias}} + \gamma_{i}^{\alpha} \]

- integer-valued cost

-0.6 <= in1 <= 9.55
-4.5 <= in2 <= 0.2
-0.06 <= in3 <= 2.11
-0.3 <= in4 <= 1.51
Key Idea: Quantization for efficiency is an optimization problem!

\[
\begin{align*}
&\text{minimize: } \gamma = \sum_{i=1}^{n} \gamma_{i}^{\text{dot}} + \gamma_{i}^{\text{bias}} + \gamma_{i}^{\alpha} \\
&\text{subject to: } \epsilon_{n} \leq \epsilon_{\text{target}}
\end{align*}
\]

- integer-valued cost
- real-valued error constraint

\[-0.6 \leq \text{in}_1 \leq 9.55 \]
\[-4.5 \leq \text{in}_2 \leq 0.2 \]
\[-0.06 \leq \text{in}_3 \leq 2.11 \]
\[-0.3 \leq \text{in}_4 \leq 1.51 \]
Quantization Formulated as an Optimization Problem

minimize: $\gamma = \sum_{i=1}^{n} \gamma_i^{\text{dot}} + \gamma_i^{\text{bias}} + \gamma_i^{\alpha}$

subject to:

$\epsilon_n \leq \epsilon_{\text{target}}$

• integer-valued cost
• real-valued error constraint
Quantization in Fixed-Point Precision

mixed-integer problem

minimize: \( \gamma = \sum_{i=1}^{n} \gamma_i^{\text{dot}} + \gamma_i^{\text{bias}} + \gamma_i^{\alpha} \)

subject to:
\( \epsilon_n \leq \epsilon_{\text{target}} \)

fixed-point representation
Quantization in Fixed-Point Precision

**minimize:** $\gamma = \sum_{i=1}^{n} \gamma_i^{\text{dot}} + \gamma_i^{\text{bias}} + \gamma_i^{\alpha}$

**subject to:**

$\epsilon_n \leq \epsilon_{\text{target}}$

$I_i^{\text{op}} \geq \text{intBits} \left( R_i^{\text{op}} + \epsilon_i \right)$

**constraint ensuring no overflow**

fixed-point representation

integer bits

fractional bits

sign bit
mixed-integer problem

minimize: \( \gamma = \sum_{i=1}^{n} \gamma_{i}^{\text{dot}} + \gamma_{i}^{\text{bias}} + \gamma_{i}^{\alpha} \)

subject to:
\( \epsilon_{n} \leq \epsilon_{\text{target}} \)
\( I_{i}^{op} \geq \text{intBits} \left( R_{i}^{op} + \epsilon_{i} \right) \)

mixed-integer non-linear hard problem!
Quantization in Fixed-Point Precision

mixed-integer problem

minimize: \( \gamma = \sum_{i=1}^{n} \gamma_{i}^{\text{dot}} + \gamma_{i}^{\text{bias}} + \gamma_{i}^{\alpha} \)

subject to:
\[ \epsilon_n \leq \epsilon_{\text{target}} \]
\[ I_{i}^{\text{op}} \geq \text{intBits} \left( R_{i}^{\text{op}} + \epsilon_{i} \right) \]

mixed-integer non-linear hard problem!
mixed-integer problem

\[
\begin{align*}
\text{minimize: } & \quad \gamma = \sum_{i=1}^{n} \gamma_i^{\text{dot}} + \gamma_i^{\text{bias}} + \gamma_i^{\alpha} \\
\text{subject to: } & \quad \epsilon_n \leq \epsilon_{\text{target}} \\
& \quad I_{i}^{\text{op}} \geq \text{intBits} \left( R_{i}^{\text{op}} + \epsilon_i \right)
\end{align*}
\]

mixed-integer non-linear hard problem!

Our Idea: Reduce to Mixed Integer Linear Programming (MILP) Problem!
Overview: Reduction to MILP

\[
\text{minimize: } \gamma = \sum_{i=1}^{n} \gamma_i^{\text{dot}} + \gamma_i^{\text{bias}} + \gamma_i^{\alpha}
\]

subject to:

\[
\epsilon_n \leq \epsilon_{\text{target}}
\]

\[
I_{i_{\text{op}}}^{\text{op}} \geq \text{intBits}\left(R_{i_{\text{op}}}^{\text{op}} + \epsilon_i\right)
\]

fixed-point representation

\[
\text{integer bits} \quad \text{fractional bits}
\]

sign bit
Overview: Reduction to MILP

minimize: \( \gamma = \sum_{i=1}^{n} \gamma_i^{\text{dot}} + \gamma_i^{\text{bias}} + \gamma_i^{\alpha} \)

subject to:
\( \epsilon_n \leq \epsilon_{\text{target}} \)
\( I_i^{\text{op}} \geq \text{intBits} \left( R_i^{\text{op}} + \epsilon_i \right) \)

over-approximate integer bits separately

**Fixed-point representation**

- **Sign bit**
- **Integer bits**
- **Fractional bits**
minimize: \( \gamma = \sum_{i=1}^{n} \gamma_i^{\text{dot}} + \gamma_i^{\text{bias}} + \gamma_i^{\alpha} \)

subject to:

\[ \varepsilon_n \leq \varepsilon_{\text{target}} \]

\[ I_i^{op} \geq \text{intBits} \left( R_i^{op} + \epsilon_i \right) \]

• over-approximate integer bits separately
Overview: Reduction to MILP

\[
\text{minimize: } \gamma = \sum_{i=1}^{n} \gamma_i^{\text{dot}} + \gamma_i^{\text{bias}} + \gamma_i^{\alpha} \\
\text{subject to: } \epsilon_n \leq \epsilon_{\text{target}} \\
I_i^{\text{op}} \geq \text{intBits}\left(R_i^{\text{op}} + \epsilon_i\right)
\]

- over-approximate integer bits separately
- linearize bias cost and error constraint exactly
Overview: Reduction to MILP

minimize: \( \gamma = \sum_{i=1}^{n} \gamma_{i}^{\text{dot}} + \gamma_{i}^{\text{bias}} + \gamma_{i}^{\alpha} \)

subject to:
\( \epsilon_{n} \leq \epsilon_{\text{target}} \)
\( I_{i}^{\text{op}} \geq \text{intBits} \left( R_{i}^{\text{op}} + \epsilon_{i} \right) \)

- over-approximate integer bits separately
- linearize bias cost and error constraint exactly
- abstract dot product
Overview: Reduction to MILP

minimize: \( \gamma = \sum_{i=1}^{n} \gamma_{i}^{\text{dot}} + \gamma_{i}^{\text{bias}} + \gamma_{i}^{\alpha} \)

subject to:
\( \epsilon_n \leq \epsilon_{\text{target}} \)
\( I_{i}^{\text{op}} \geq \text{intBits} \left( R_{i}^{\text{op}} + \epsilon_i \right) \)

• over-approximate integer bits separately
• linearize bias cost and error constraint exactly
• abstract dot product
Linearization Step 1: Computing Integer Bits

step 1: computing integer bits using interval arithmetic
Linearization Step 1: Computing Integer Bits

weights1

\[
\begin{bmatrix}
-0.6, 9.55 \\
-4.5, 0.2 \\
-0.06, 2.11 \\
-0.3, 1.51
\end{bmatrix}
\]

\[
\begin{bmatrix}
\cdots
\end{bmatrix}
\]

\[\begin{bmatrix}
-0.034, -0.01, -0.13, 0.04 \\
0.13, 0.01, 0.05, -0.17
\end{bmatrix}\]

compute integer bits for variables

-0.6 \leq in1 \leq 9.55
-4.5 \leq in2 \leq 0.2
-0.06 \leq in3 \leq 2.11
-0.3 \leq in4 \leq 1.51

res +/- \text{1e-3}
Linearization Step 1: Computing Integer Bits

\[-0.6 \leq \text{in1} \leq 9.55\]
\[-4.5 \leq \text{in2} \leq 0.2\]
\[-0.06 \leq \text{in3} \leq 2.11\]
\[-0.3 \leq \text{in4} \leq 1.51\]

\[
\begin{bmatrix}
-0.6, 9.55 \\
-4.5, 0.2 \\
-0.06, 2.11 \\
-0.3, 1.51
\end{bmatrix}
\]

\[
\begin{bmatrix}
-0.034, -0.01, -0.13, 0.04 \\
0.13, 0.1, 0.05, -0.17
\end{bmatrix}
\]

\[\text{relu} \]

\[\text{out} \]

\[\text{res} +/− 1e-3\]
Linearization Step 1: Computing Integer Bits

Computed integer bits for all variables and constants without overflow
Overview: Reduction to MILP

minimize: \( \gamma = \sum_{i=1}^{n} \gamma_{i}^{\text{dot}} + \gamma_{i}^{\text{bias}} + \gamma_{i}^{\alpha} \)

subject to:
\[ \epsilon_n \leq \epsilon_{\text{target}} \]
\[ I_{i}^{\text{op}} \geq \text{intBits} \left( R_{i}^{\text{op}} + \epsilon_{i} \right) \]

over-approximate integer bits separately

- linearize bias cost and error constraint exactly
- abstract dot product
Linearization Step 2: Exact Linearization of Cost

\[
\begin{align*}
\text{minimize:} \quad \gamma &= \sum_{i=1}^{n} \gamma_i^{dot} + \gamma_i^{bias} + \gamma_i^\alpha \\
\text{subject to:} \\
\epsilon_n &\leq \epsilon_{target} \\
I_i^{op} &\geq \text{intBits} \left( R_i^{op} + \epsilon_i \right)
\end{align*}
\]

\[\gamma_i^{bias} = \max(\pi_i^{dot}, \pi_i^{bias})\]

non-linear function
Linearization Step 2: Exact Linearization of Cost

Minimize: \( \gamma = \sum_{i=1}^{n} \gamma_{i}^{\text{dot}} + \gamma_{i}^{\text{bias}} + \gamma_{i}^{\alpha} \)

Subject to:
\[
\epsilon_n \leq \epsilon_{\text{target}}
\]
\[
I_{i}^{\text{op}} \geq \text{intBits} \left( R_{i}^{\text{op}} + \epsilon_{i} \right)
\]

\( \gamma_{i}^{\text{bias}} = \max(\pi_{i}^{\text{dot}}, \pi_{i}^{\text{bias}}) \)

\[ \text{c1: } \gamma_{i}^{\text{bias}} \geq \pi_{i}^{\text{dot}} \]
\[ \text{c2: } \gamma_{i}^{\text{bias}} \geq \pi_{i}^{\text{bias}} \]
Overview: Reduction to MILP

minimize: \( \gamma = \sum_{i=1}^{n} \gamma_i^{dot} + \gamma_i^{bias} + \gamma_i^{\alpha} \)

subject to:
\[ \epsilon_n \leq \epsilon_{target} \]
\[ I_{i}^{op} \geq \text{intBits} \left( R_{i}^{op} + \epsilon_i \right) \]

- over-approximate integer bits separately
- linearize bias cost and error constraint exactly
- abstract dot product
Linearization Step 3: Abstract Dot Product

Assume a precision for weights, correct it later.

\[
\begin{align*}
\text{minimize: } \gamma &= \sum_{i=1}^{n} \gamma_i^{\text{dot}} + \gamma_i^{\text{bias}} + \gamma_i^{\alpha} \\
\text{subject to: } \epsilon_n &\leq \epsilon_{\text{target}} \\
I_i^{\text{op}} &\geq \text{intBits} \left( R_i^{\text{op}} + \epsilon_i \right)
\end{align*}
\]

- Over-approximate integer bits separately
- Linearize bias cost and error constraint exactly
- Abstract dot product
Optimizing Fractional Bits for Dot and Bias Products

Linearized Problem

minimize: \( \gamma = \sum_{i=1}^{n} \gamma_{i}^{\text{dot}} + \gamma_{i}^{\text{bias}} + \gamma_{i}^{\alpha} \)

subject to:

\( \epsilon_n \leq \epsilon_{\text{target}} \)

\( I_{i}^{\text{op}} \geq \text{intBits} \left( R_{i}^{\text{op}} + \epsilon_{i} \right) \)
Optimizing Fractional Bits for Dot and Bias Products

Linearized Problem

minimize: \( \gamma = \sum_{i=1}^{n} \gamma_i^{\text{dot}} + \gamma_i^{\text{bias}} + \gamma_i^{\alpha} \)

subject to:

\( \epsilon_n \leq \epsilon_{\text{target}} \)

\( I_i^{op} \geq \text{intBits} \left( R_i^{op} + \epsilon_i \right) \)
• computed **integer bits** for all variables and constants
• optimized **fractional bits** for dot and bias results assuming precision of weights
Assign Correctly Rounded Precision to Weights

- computed integer bits for all variables and constants
- optimized fractional bits for dot and bias results assuming precision of weights
- assigning correctly rounded precision for all variables and constants
Assign Correctly Rounded Precision to Weights

- computed integer bits for all variables and constants
- optimized fractional bits for dot and bias results assuming precision of weights
- assigning correctly rounded precision for all variables and constants

* A Correctly-Rounded Fixed-Point-Arithmetic DotProduct Algorithm, Sylvie Boldo, Diane Gallois-Wong, and Thibault Hilaire, ARITH 2020
Aster: Sound Quantizer for NN Controllers

```python
def UnicycleController(in: Vector): Vector = {
    weights1 = Matrix[...]
    weights2 = Matrix[...]
    bias1 = Vector(…)
    bias2 = Vector(...)
    x1 = relu(weights1 * in + bias1)
    out = linear(weights2 * x1 + bias2)
    return out
}
ensuring (res +/- 1e-3)
```

high-level model
Aster: Sound Quantizer for NN Controllers

#include <math.h>
#include <ap_fixed.h>
#include <hls_math.h>
#include <ap_fixed.h>

void nn1(ap_fixed<24,5> x_0, ap_fixed<24,4> x_1, ap_fixed<24,3> x_2,
ap_fixed<24,2> x_3, ap_fixed<27,8> _result[2]) {
    ap_fixed<24,1> weights1_0_0 = -0.036691424;
    ...
    ap_fixed<27,8> layer2_dot_1 = (_tmp4994 + _tmp4995);
    ap_fixed<27,8> layer2_bias_0 = (layer2_dot_0 + (ap_fixed<27,13> (bias2_0));
    ap_fixed<27,8> layer2_bias_1 = (layer2_dot_1 + (ap_fixed<27,13> (bias2_1));
    ap_fixed<27,8> layer2_0 = (layer2_bias_0);
    ap_fixed<27,8> layer2_1 = (layer2_bias_1);
    _result[0] = layer2_0;
    _result[1] = layer2_1;
}

def UnicycleController(in: Vector): Vector = {
    weights1 = Matrix[...]
    weights2 = Matrix[...]
    bias1 = Vector(...)
    bias2 = Vector(...)
    x1 = relu(weights1 * in + bias1)
    out = linear(weights2 * x1 + bias2)
    return out
} ensuring (res +/- 1e-3)

mixed-precision fixed-point code

#include <math.h>
#include <ap_fixed.h>
#include <hls_math.h>
#include <ap_fixed.h>

void nn1(ap_fixed<24,5> x_0, ap_fixed<24,4> x_1, ap_fixed<24,3> x_2,
ap_fixed<24,2> x_3, ap_fixed<27,8> _result[2]) {
    ap_fixed<24,1> weights1_0_0 = -0.036691424;
    ...
    ap_fixed<27,8> layer2_dot_1 = (_tmp4994 + _tmp4995);
    ap_fixed<27,8> layer2_bias_0 = (layer2_dot_0 + (ap_fixed<27,13> (bias2_0));
    ap_fixed<27,8> layer2_bias_1 = (layer2_dot_1 + (ap_fixed<27,13> (bias2_1));
    ap_fixed<27,8> layer2_0 = (layer2_bias_0);
    ap_fixed<27,8> layer2_1 = (layer2_bias_1);
    _result[0] = layer2_0;
    _result[1] = layer2_1;
}

high-level model
Aster: Sound Quantizer for NN Controllers

```python
def UnicycleController(in: Vector): Vector = {
    require(-0.6<=in1<=9.55 && -4.5<=in2<=0.2 && -0.06<=in3<=2.11 && -0.3<=in4<=1.51)
    weights1 = Matrix[...]
    weights2 = Matrix[...]
    bias1 = Vector[...]
    bias2 = Vector[...]
    x1 = relu(weights1 * in + bias1)
    out = linear(weights2 * x1 + bias2)
    return out
}
ensuring (res +/- 1e-3)
```

mixed-precision fixed-point code

```c
#include <math.h>
#include <ap_fixed.h>
#include <hls_math.h>
#include <ap_fixed.h>

void nn1(ap_fixed<24,5> x_0, ap_fixed<24,4> x_1, ap_fixed<24,3> x_2, ap_fixed<24,2> x_3, ap_fixed<27,8> _result[2]) {
    ap_fixed<24,1> weights1_0_0 = -0.036691424;
    ...
    ap_fixed<24,1> layer2_dot_1 = (_tmp4994 + _tmp4995);
    ap_fixed<27,8> layer2_dot_0 = (layer2_dot_0 + (ap_fixed<24,3> x_2, ap_fixed<24,2> x_3, ap_fixed<27,8> _result[2]) {
        ap_fixed<24,1> weights1_0_0 = -0.036691424;
    ...
    ap_fixed<27,8> layer2_0 = (layer2_0 + (ap_fixed<24,3> x_2, ap_fixed<24,2> x_3, ap_fixed<27,8> _result[2]) {
        ap_fixed<24,1> weights1_0_0 = -0.036691424;
    ...
    ap_fixed<27,8> layer2_1 = (layer2_1 + (ap_fixed<24,3> x_2, ap_fixed<24,2> x_3, ap_fixed<27,8> _result[2]) {
        ap_fixed<24,1> weights1_0_0 = -0.036691424;
    ...
    _result[0] = layer2_0;
    _result[1] = layer2_1;
}
```

high-level model
directly compiled
## Aster vs State-of-the-Art in terms of Latency

<table>
<thead>
<tr>
<th>benchmarks</th>
<th>#params</th>
<th>Daisy</th>
<th>Aster</th>
</tr>
</thead>
<tbody>
<tr>
<td>InvPendulum</td>
<td>60</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MountainCar</td>
<td>336</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MPC</td>
<td>720</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DoublePendulum</td>
<td>825</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ACC3</td>
<td>980</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Unicycle</td>
<td>3,500</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Airplane</td>
<td>13,540</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ControllerTora</td>
<td>20,800</td>
<td></td>
<td></td>
</tr>
<tr>
<td>AC8</td>
<td>44,545</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Latencies of implementations considering target error 1e-3
### Aster vs State-of-the-Art in terms of Latency

<table>
<thead>
<tr>
<th>benchmarks</th>
<th>#params</th>
<th>Daisy</th>
<th>Aster</th>
</tr>
</thead>
<tbody>
<tr>
<td>InvPendulum</td>
<td>60</td>
<td>12</td>
<td></td>
</tr>
<tr>
<td>MountainCar</td>
<td>336</td>
<td>27</td>
<td></td>
</tr>
<tr>
<td>MPC</td>
<td>720</td>
<td>inf</td>
<td></td>
</tr>
<tr>
<td>DoublePendulum</td>
<td>825</td>
<td>36</td>
<td></td>
</tr>
<tr>
<td>ACC3</td>
<td>980</td>
<td>49</td>
<td></td>
</tr>
<tr>
<td>Unicycle</td>
<td>3,500</td>
<td>178</td>
<td></td>
</tr>
<tr>
<td>Airplane</td>
<td>13,540</td>
<td>TO</td>
<td></td>
</tr>
<tr>
<td>ControllerTora</td>
<td>20,800</td>
<td>TO</td>
<td></td>
</tr>
<tr>
<td>AC8</td>
<td>44,545</td>
<td>TO</td>
<td></td>
</tr>
</tbody>
</table>

Latencies of implementations considering target error $1e^{-3}$, **TO**: timed out after 5 hours, **inf**: tool returns infeasible.
Aster vs State-of-the-Art in terms of Latency

<table>
<thead>
<tr>
<th>benchmarks</th>
<th>#params</th>
<th>Daisy</th>
<th>Aster</th>
</tr>
</thead>
<tbody>
<tr>
<td>InvPendulum</td>
<td>60</td>
<td>12</td>
<td>12</td>
</tr>
<tr>
<td>MountainCar</td>
<td>336</td>
<td>27</td>
<td>25</td>
</tr>
<tr>
<td>MPC</td>
<td>720</td>
<td>inf</td>
<td>35</td>
</tr>
<tr>
<td>DoublePendulum</td>
<td>825</td>
<td>36</td>
<td>27</td>
</tr>
<tr>
<td>ACC3</td>
<td>980</td>
<td>49</td>
<td>44</td>
</tr>
<tr>
<td>Unicycle</td>
<td>3,500</td>
<td>178</td>
<td>27</td>
</tr>
<tr>
<td>Airplane</td>
<td>13,540</td>
<td>TO</td>
<td></td>
</tr>
<tr>
<td>ControllerTora</td>
<td>20,800</td>
<td>TO</td>
<td></td>
</tr>
<tr>
<td>AC8</td>
<td>44,545</td>
<td>TO</td>
<td></td>
</tr>
</tbody>
</table>

Latencies of implementations considering target error 1e-3, **TO**: timed out after 5 hours, **inf**: tool returns infeasible
### Aster vs State-of-the-Art in terms of Latency

<table>
<thead>
<tr>
<th>benchmarks</th>
<th>#params</th>
<th>Daisy</th>
<th>Aster</th>
</tr>
</thead>
<tbody>
<tr>
<td>InvPendulum</td>
<td>60</td>
<td>12</td>
<td>12</td>
</tr>
<tr>
<td>MountainCar</td>
<td>336</td>
<td>27</td>
<td>25</td>
</tr>
<tr>
<td>MPC</td>
<td>720</td>
<td>inf</td>
<td>35</td>
</tr>
<tr>
<td>DoublePendulum</td>
<td>825</td>
<td>36</td>
<td>27</td>
</tr>
<tr>
<td>ACC3</td>
<td>980</td>
<td>49</td>
<td>44</td>
</tr>
<tr>
<td>Unicycle</td>
<td>3,500</td>
<td>178</td>
<td>27</td>
</tr>
<tr>
<td>Airplane</td>
<td>13,540</td>
<td>TO</td>
<td></td>
</tr>
<tr>
<td>ControllerTora</td>
<td>20,800</td>
<td>TO</td>
<td></td>
</tr>
<tr>
<td>AC8</td>
<td>44,545</td>
<td>TO</td>
<td></td>
</tr>
</tbody>
</table>

Latencies of implementations considering target error 1e-3, **TO**: timed out after 5 hours, **inf**: tool returns infeasible.

Unrolled programs are too large!  

40 KLOC  
62 KLOC  
134 KLOC
Aster vs State-of-the-Art in terms of Latency

<table>
<thead>
<tr>
<th>benchmarks</th>
<th>#params</th>
<th>Daisy</th>
<th>Aster</th>
</tr>
</thead>
<tbody>
<tr>
<td>InvPendulum</td>
<td>60</td>
<td>12</td>
<td>12</td>
</tr>
<tr>
<td>MountainCar</td>
<td>336</td>
<td>27</td>
<td>25</td>
</tr>
<tr>
<td>MPC</td>
<td>720</td>
<td>inf</td>
<td>35</td>
</tr>
<tr>
<td>DoublePendulum</td>
<td>825</td>
<td>36</td>
<td>27</td>
</tr>
<tr>
<td>ACC3</td>
<td>980</td>
<td>49</td>
<td>44</td>
</tr>
<tr>
<td>Unicycle</td>
<td>3,500</td>
<td>178</td>
<td>27</td>
</tr>
<tr>
<td>Airplane</td>
<td>13,540</td>
<td>TO</td>
<td>9,001*</td>
</tr>
<tr>
<td>ControllerTora</td>
<td>20,800</td>
<td>TO</td>
<td>13,158*</td>
</tr>
<tr>
<td>AC8</td>
<td>44,545</td>
<td>TO</td>
<td>✗</td>
</tr>
</tbody>
</table>

Latencies of implementations considering target error $1e-3$, TO: timed out after 5 hours, inf: tool returns infeasible

*: compiled with explicit loops (i.e. not unrolled code), ✗: Xilinx failed to compile the implementation
## Aster vs State-of-the-Art in terms of Optimization Time

<table>
<thead>
<tr>
<th>benchmarks</th>
<th>Daisy</th>
<th>Aster</th>
</tr>
</thead>
<tbody>
<tr>
<td>InvPendulum</td>
<td>4.19s</td>
<td></td>
</tr>
<tr>
<td>MountainCar</td>
<td>43.68s</td>
<td></td>
</tr>
<tr>
<td>MPC</td>
<td></td>
<td>inf</td>
</tr>
<tr>
<td>DoublePendulum</td>
<td>4m 6.64s</td>
<td></td>
</tr>
<tr>
<td>ACC3</td>
<td>4m 52.05s</td>
<td></td>
</tr>
<tr>
<td>Unicycle</td>
<td>2h 46m 20.65s</td>
<td></td>
</tr>
<tr>
<td>Airplane</td>
<td>TO</td>
<td></td>
</tr>
<tr>
<td>ControllerTora</td>
<td>TO</td>
<td></td>
</tr>
<tr>
<td>AC8</td>
<td>TO</td>
<td></td>
</tr>
</tbody>
</table>

Optimization time averaged over 3 runs considering 1e-3 target error, **TO**: timed out after 5 hours, **inf**: tool returns infeasible.
## Aster vs State-of-the-Art in terms of Optimization Time

<table>
<thead>
<tr>
<th>benchmarks</th>
<th>Daisy</th>
<th>Aster</th>
</tr>
</thead>
<tbody>
<tr>
<td>InvPendulum</td>
<td>4.19s</td>
<td>1.66s</td>
</tr>
<tr>
<td>MountainCar</td>
<td>43.68s</td>
<td>2.22s</td>
</tr>
<tr>
<td>MPC</td>
<td>inf</td>
<td>3.50s</td>
</tr>
<tr>
<td>DoublePendulum</td>
<td>4m 6.64s</td>
<td>3.80s</td>
</tr>
<tr>
<td>ACC3</td>
<td>4m 52.05s</td>
<td>7.28s</td>
</tr>
<tr>
<td>Unicycle</td>
<td>2h 46m 20.65s</td>
<td>49.92s</td>
</tr>
<tr>
<td>Airplane</td>
<td>TO</td>
<td>17m 40.92s</td>
</tr>
<tr>
<td>ControllerTora</td>
<td>TO</td>
<td>47m 55.95s</td>
</tr>
<tr>
<td>AC8</td>
<td>TO</td>
<td>3h 49m 31.43s</td>
</tr>
</tbody>
</table>

Optimization time averaged over 3 runs considering $1e-3$ target error, **TO**: timed out after 5 hours, **inf**: tool returns infeasible
What else is there in the paper?

• The details of the MILP formulation
• Further linearization of error constraints
• Implementation details of the tool Aster
• Extensive experiments on
  - parameter evaluation of Aster
  - several benchmarks with different target errors
  - comparison of cost functions

Sound Mixed Fixed-Point Quantization of Neural Networks,
Debasmita Lohar, Clothilde Jeangoudoux, Anastasia Volkova, Eva Darulova,
ESWEEK-TECS special issue, 2023
Summary

- Optimization based mixed precision fixed-point quantization for regression models
- Quantized code guarantees target roundoff error and runs on custom hardware
- A tool Aster that is sound, automated, scalable for large networks with many parameters