Sound Mixed Fixed-Point Quantization of Neural Networks

Debasmita Lohar, Clothilde Jeangoudoux, Anastasia Volkova, Eva Darulova

EMSOFT 2023

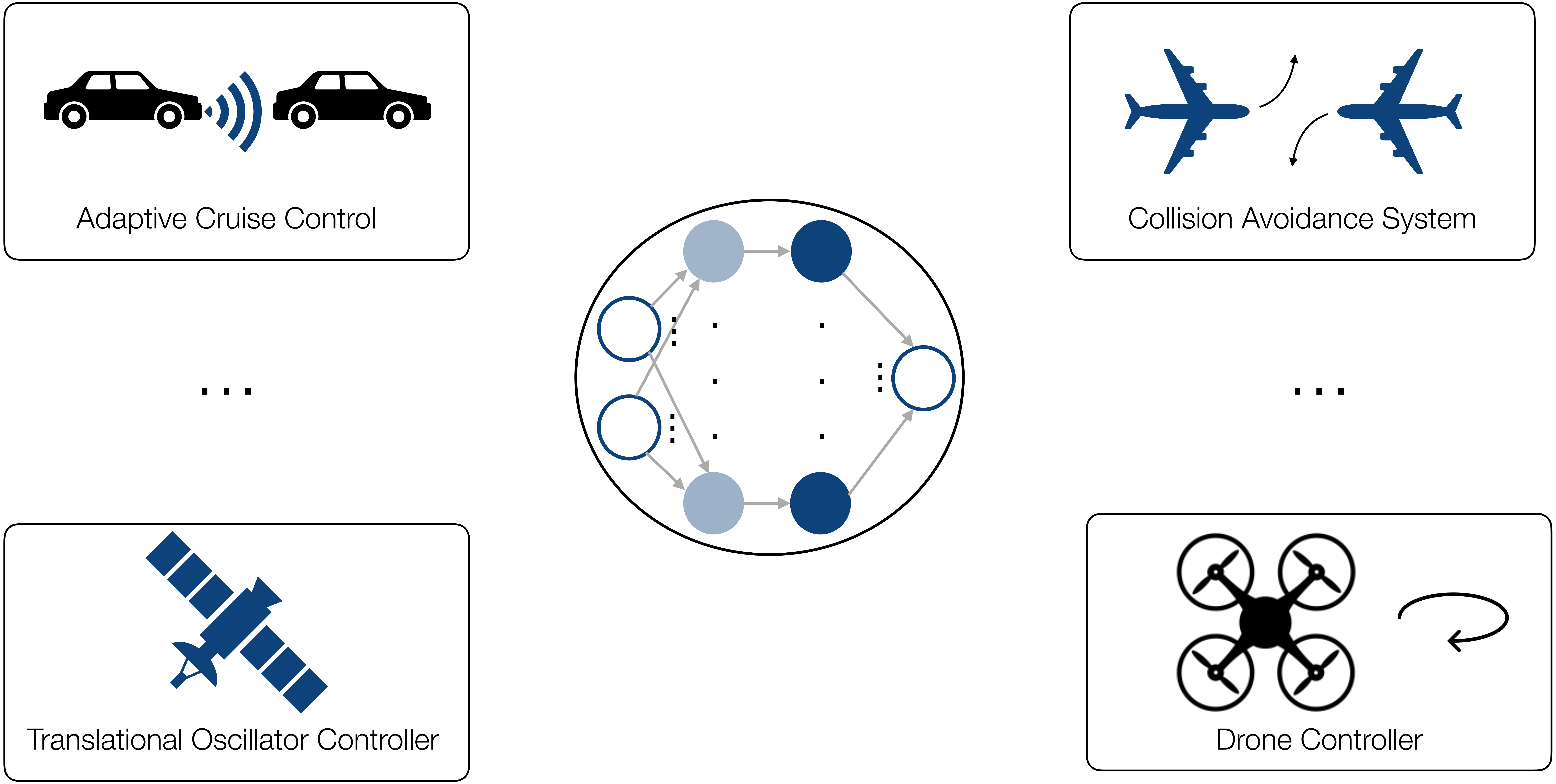
MAX PLANCK INSTITUTE
FOR SOFTWARE SYSTEMS

UNIVERSITÄT
DES
SAARLANDES

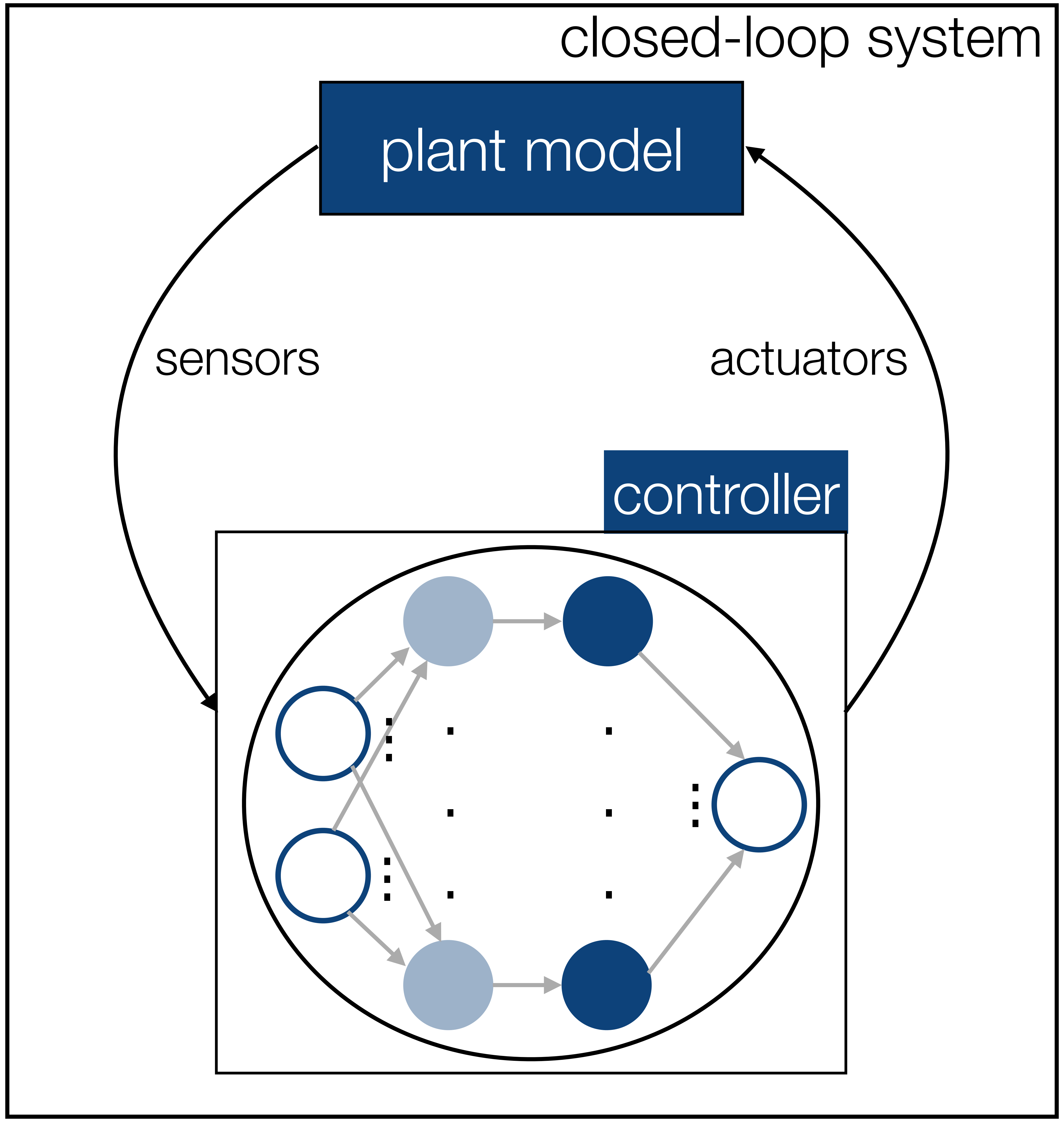
Nantes
Université

UPPSALA
UNIVERSITET

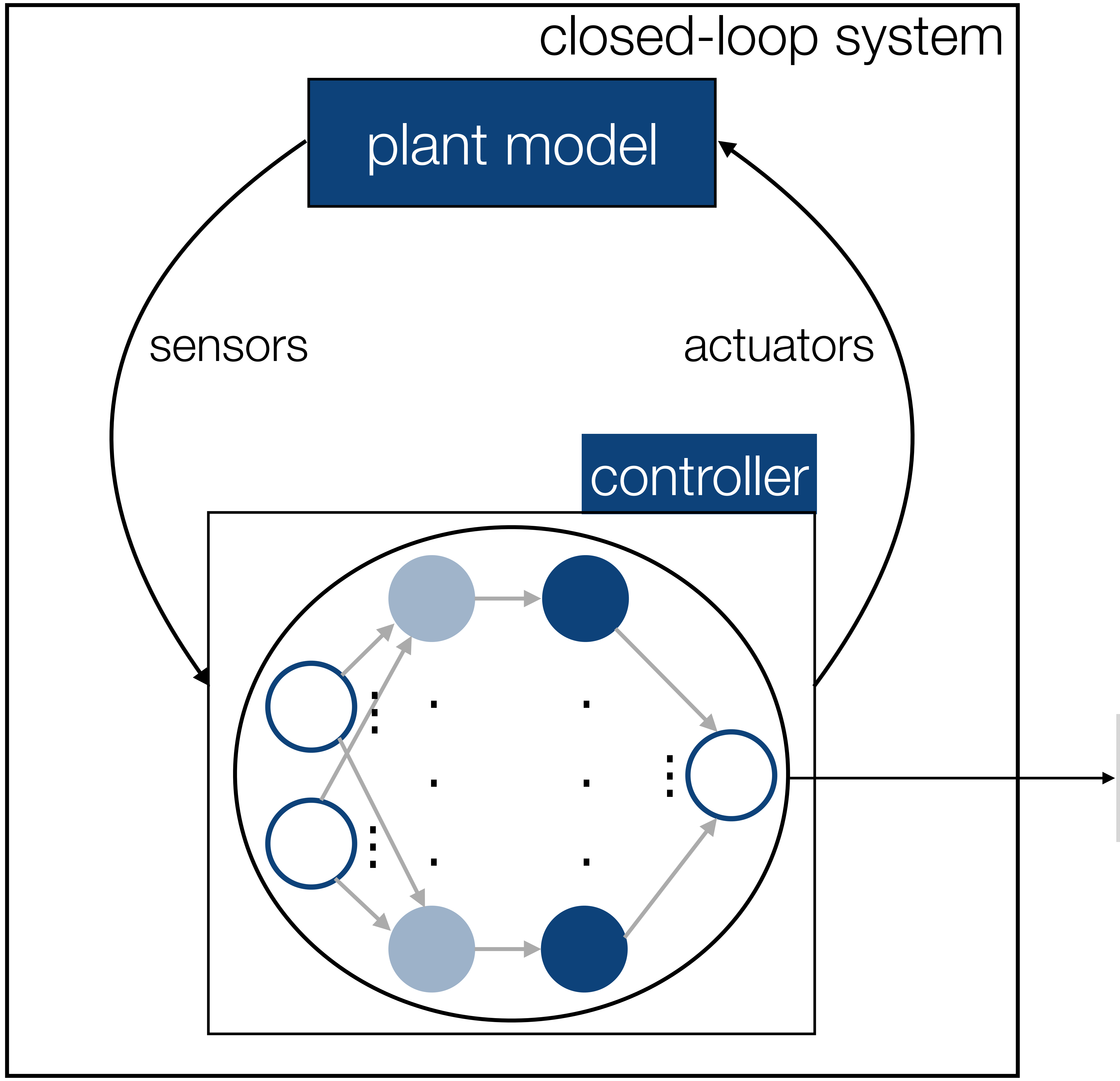
Neural networks are ubiquitous in safety-critical systems!



Neural Networks as Controllers



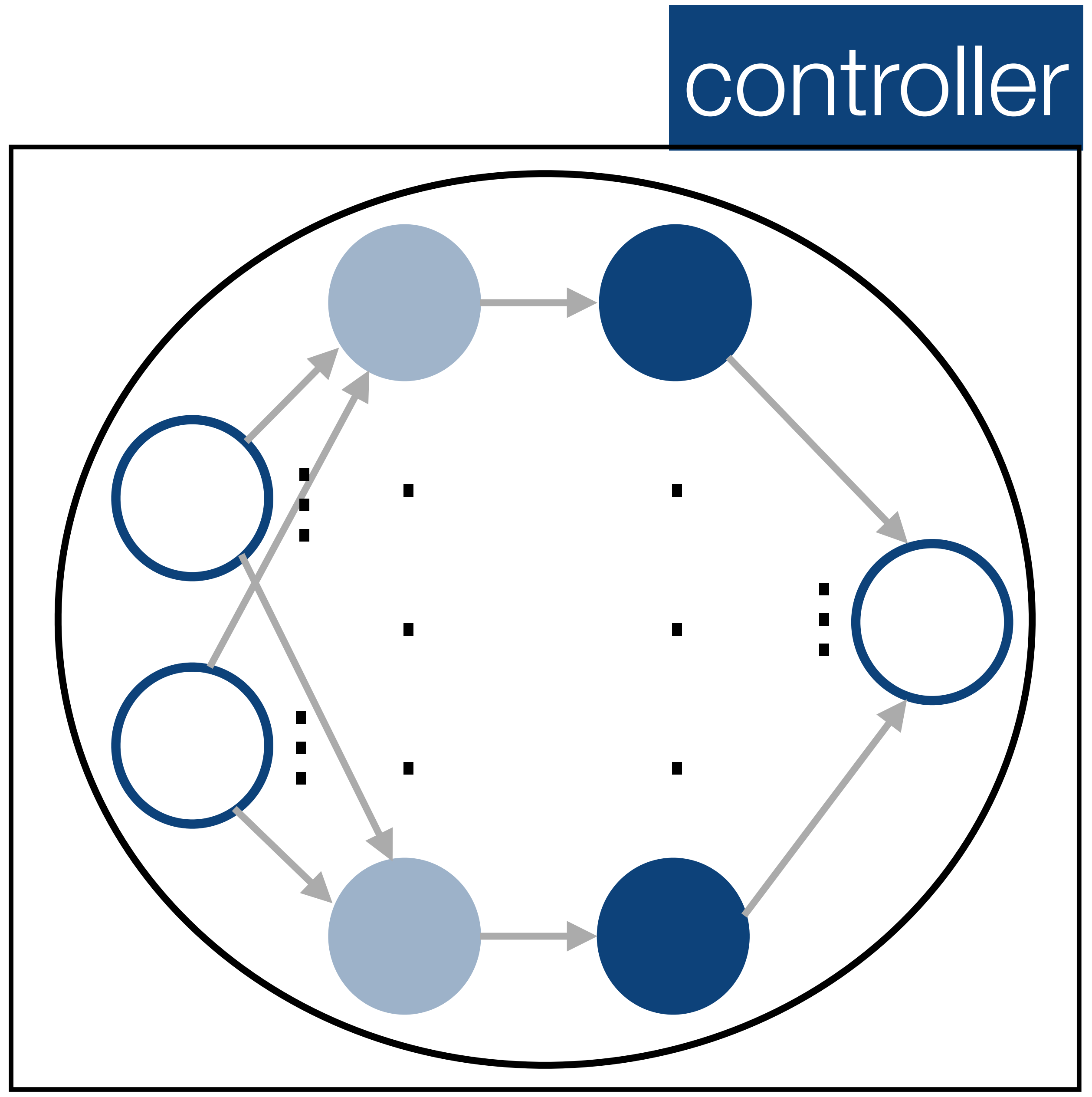
Neural Networks as Controllers



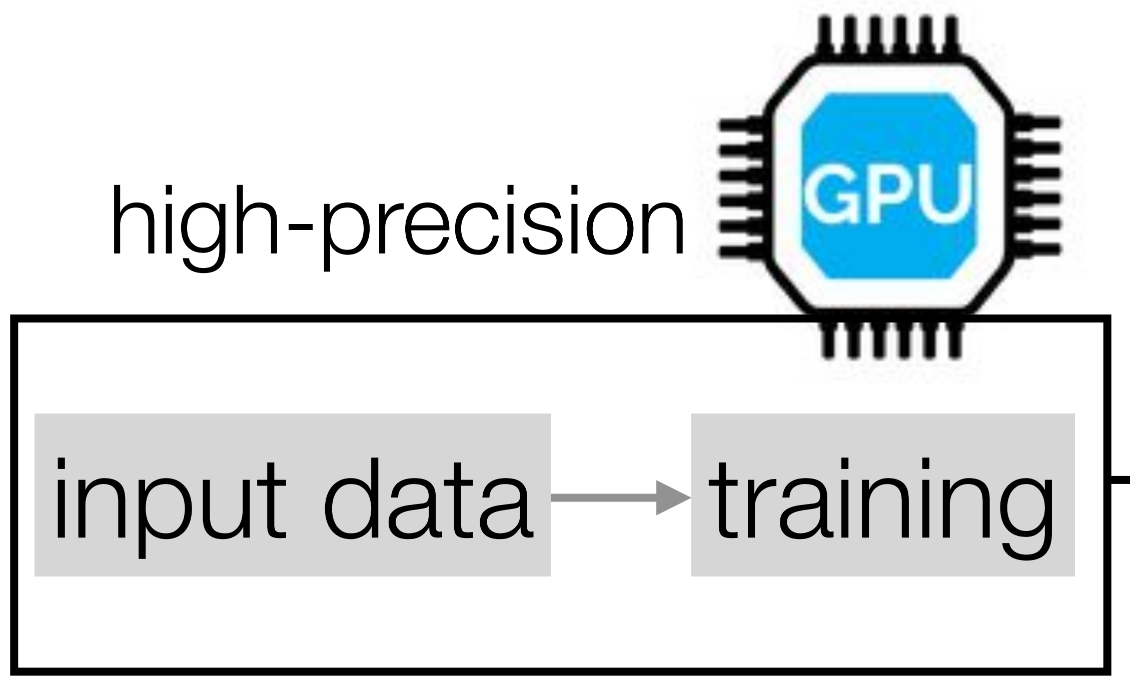

```
x1 = relu([ W1 ] x [ in ] + [ b1 ])
out = linear([ W2 ] x [ x1 ] + [ b2 ])
```

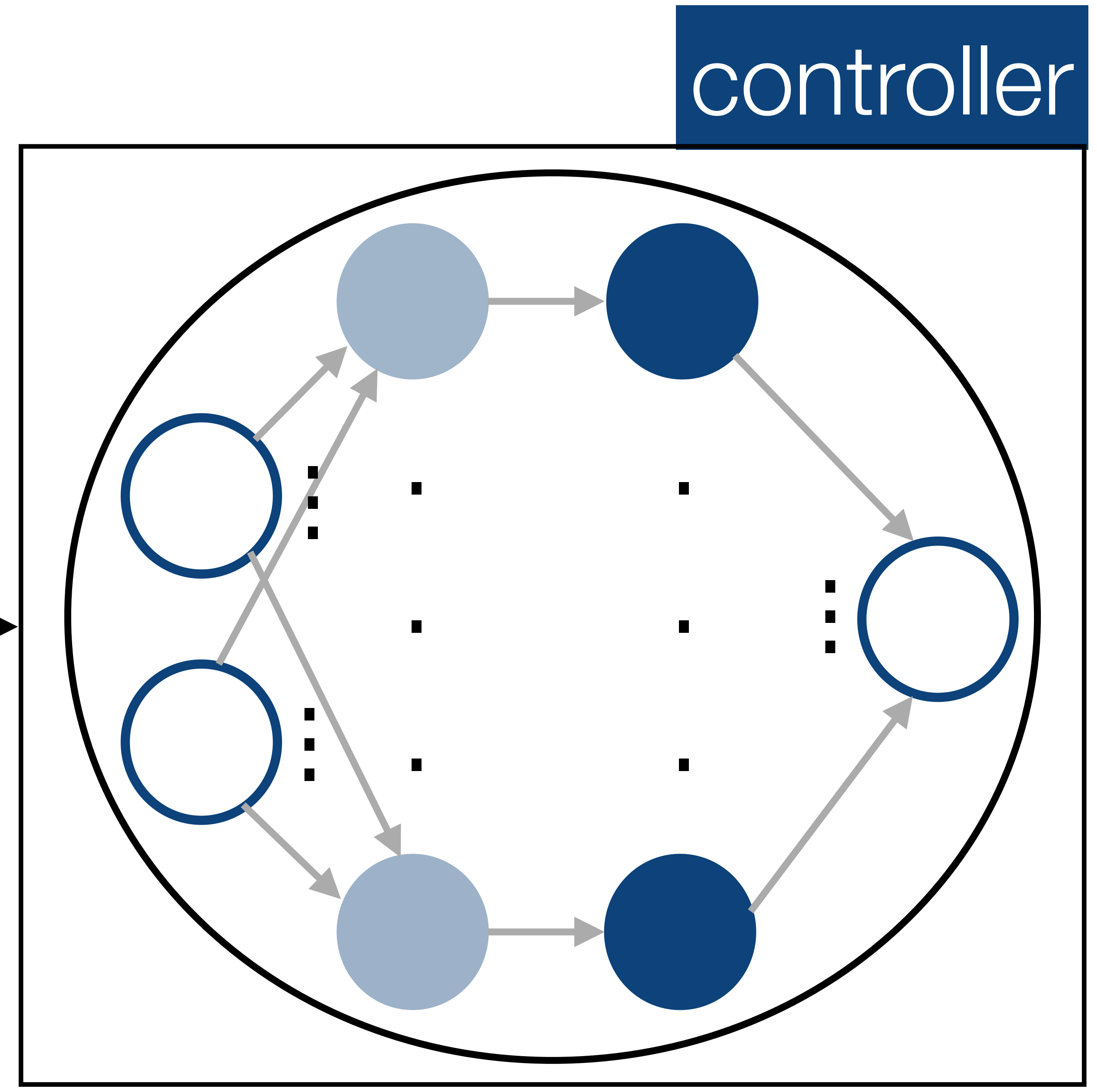
feed-forward regression model

Neural networks are usually trained in high-precision!



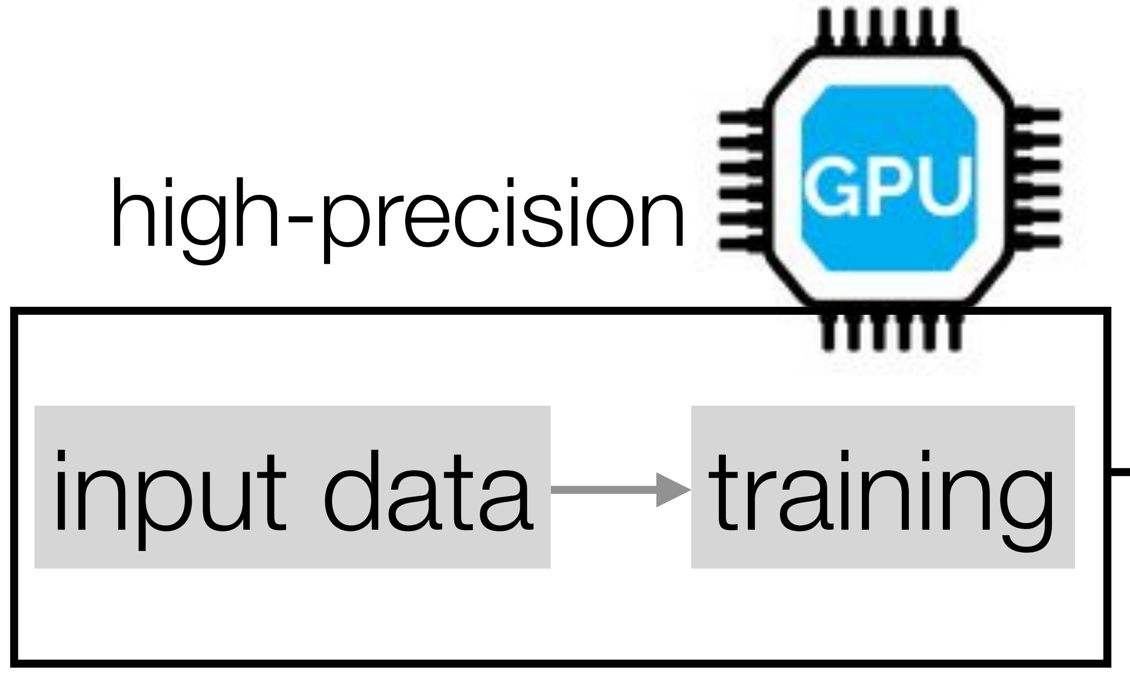
Neural networks are usually trained in high-precision!



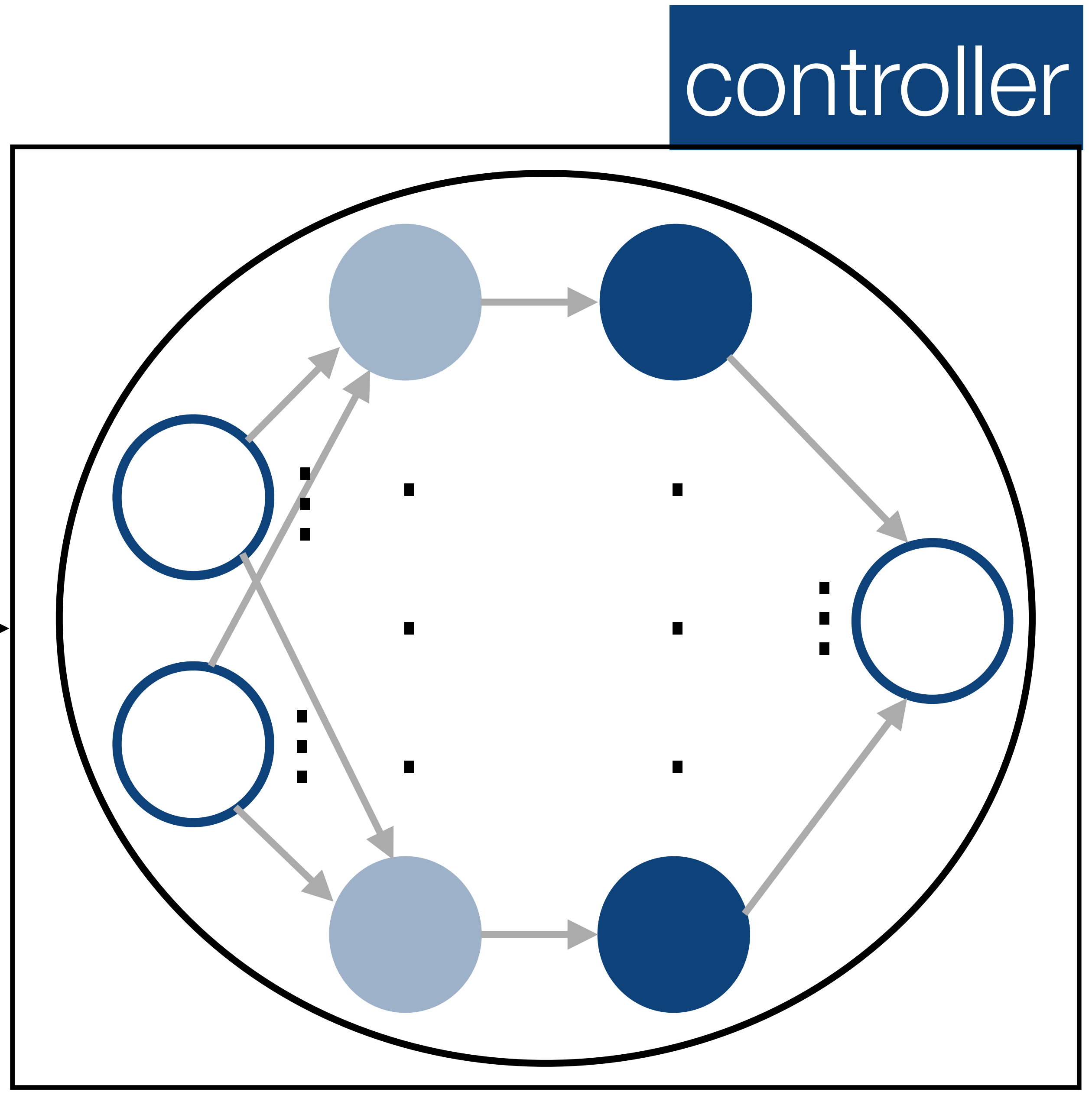


training

Model is usually in high-precision!

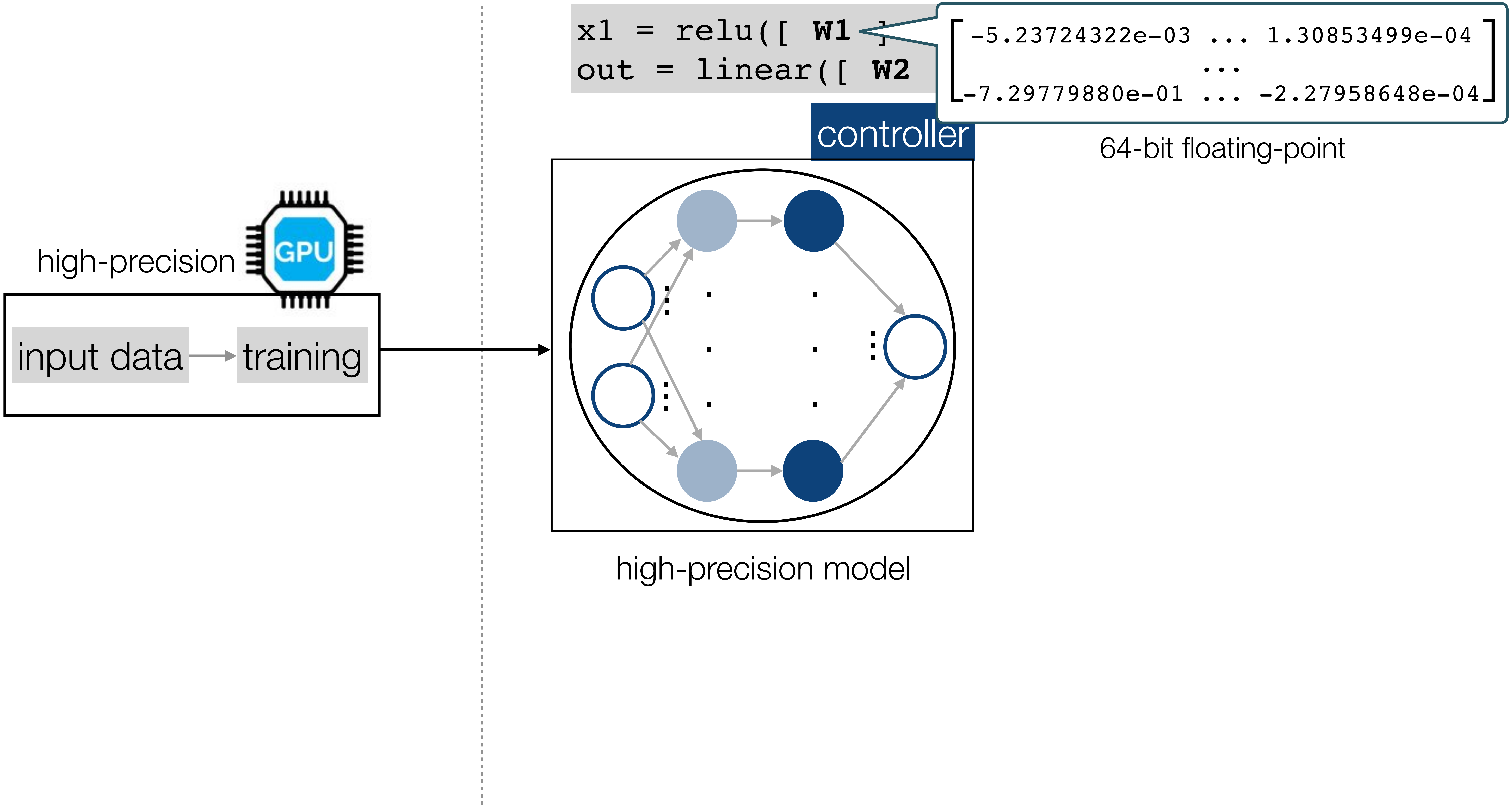



```
x1 = relu([ W1 ] x [ in ] + [ b1 ])
out = linear([ W2 ] x [ x1 ] + [ b2 ])
```

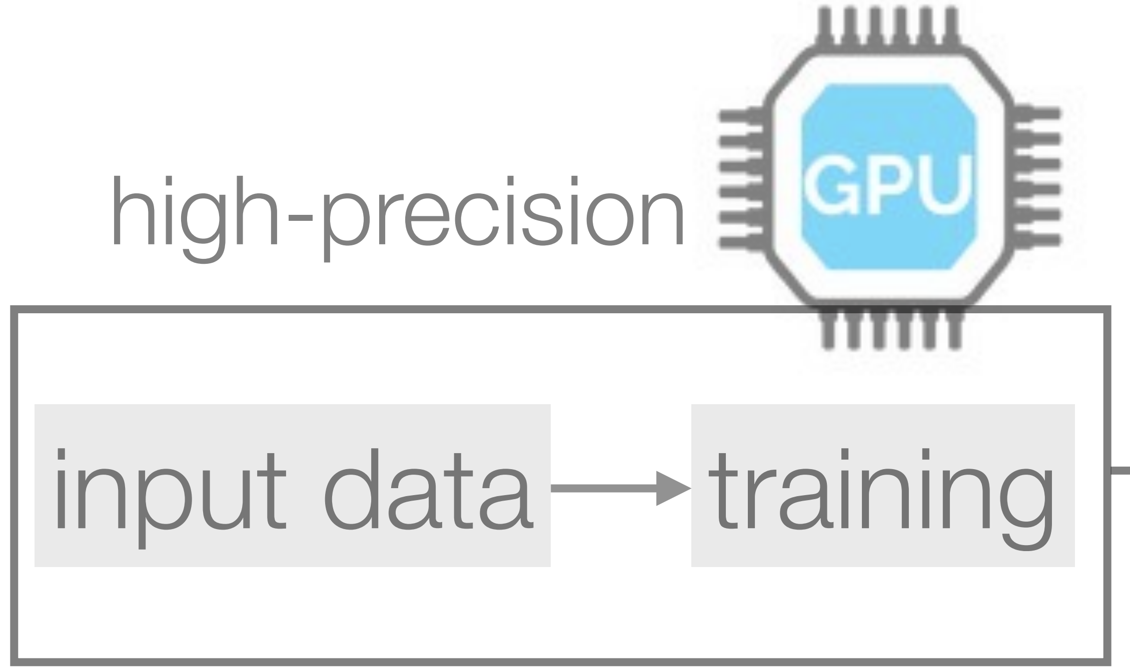


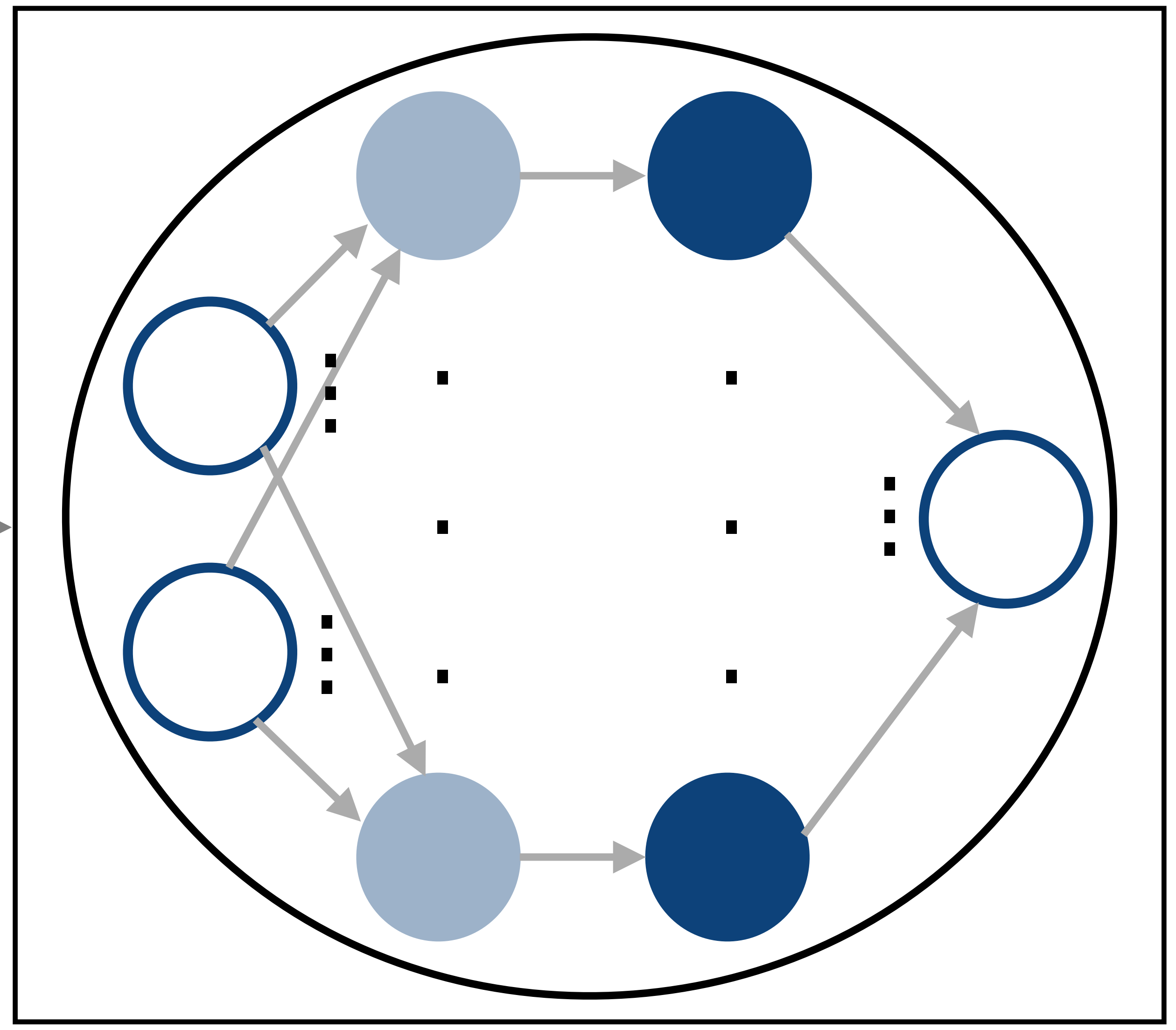
high-precision model

Model is usually in high-precision!



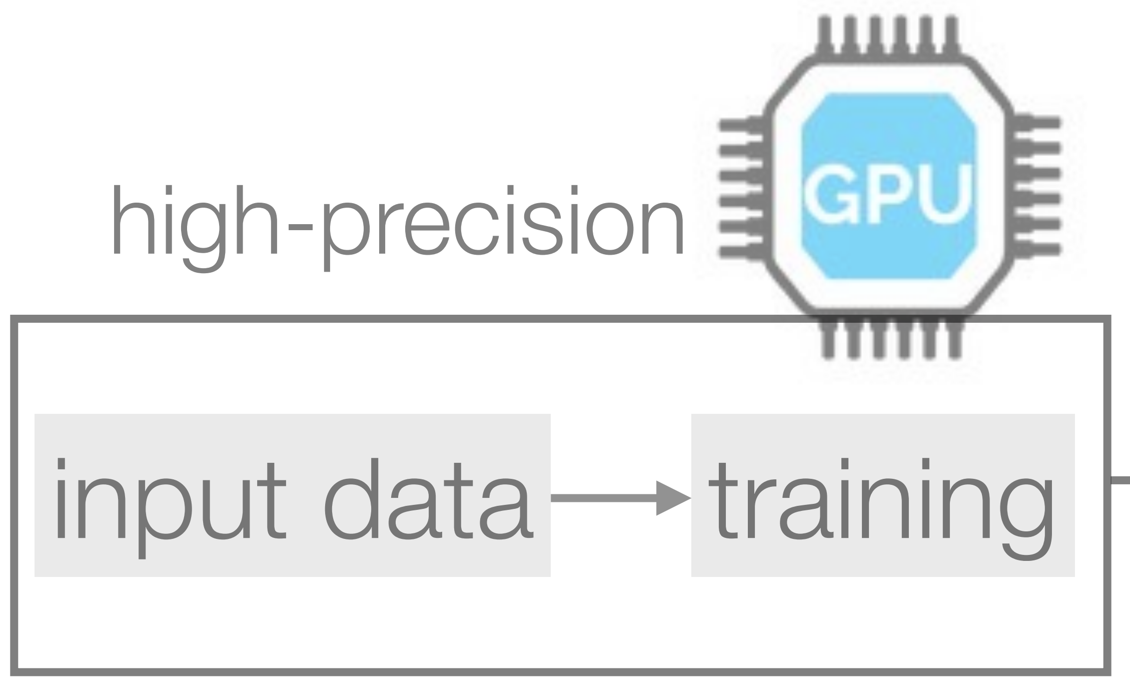
Safety Verification of Neural Network Controllers

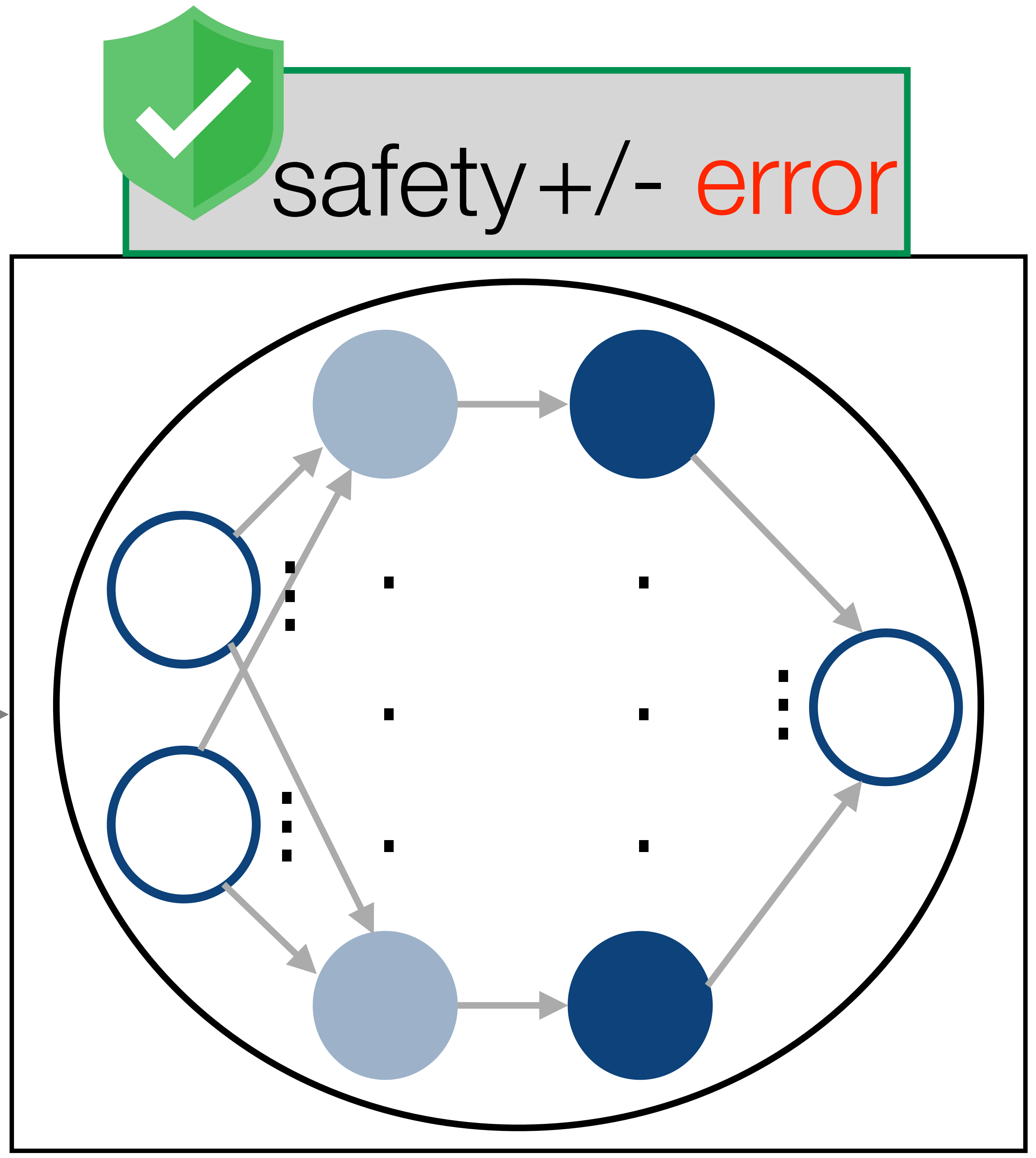




- ReachNN ’19
- Sherlock ’19
- NNV ’20
- …

Safety Verification of Neural Network Controllers

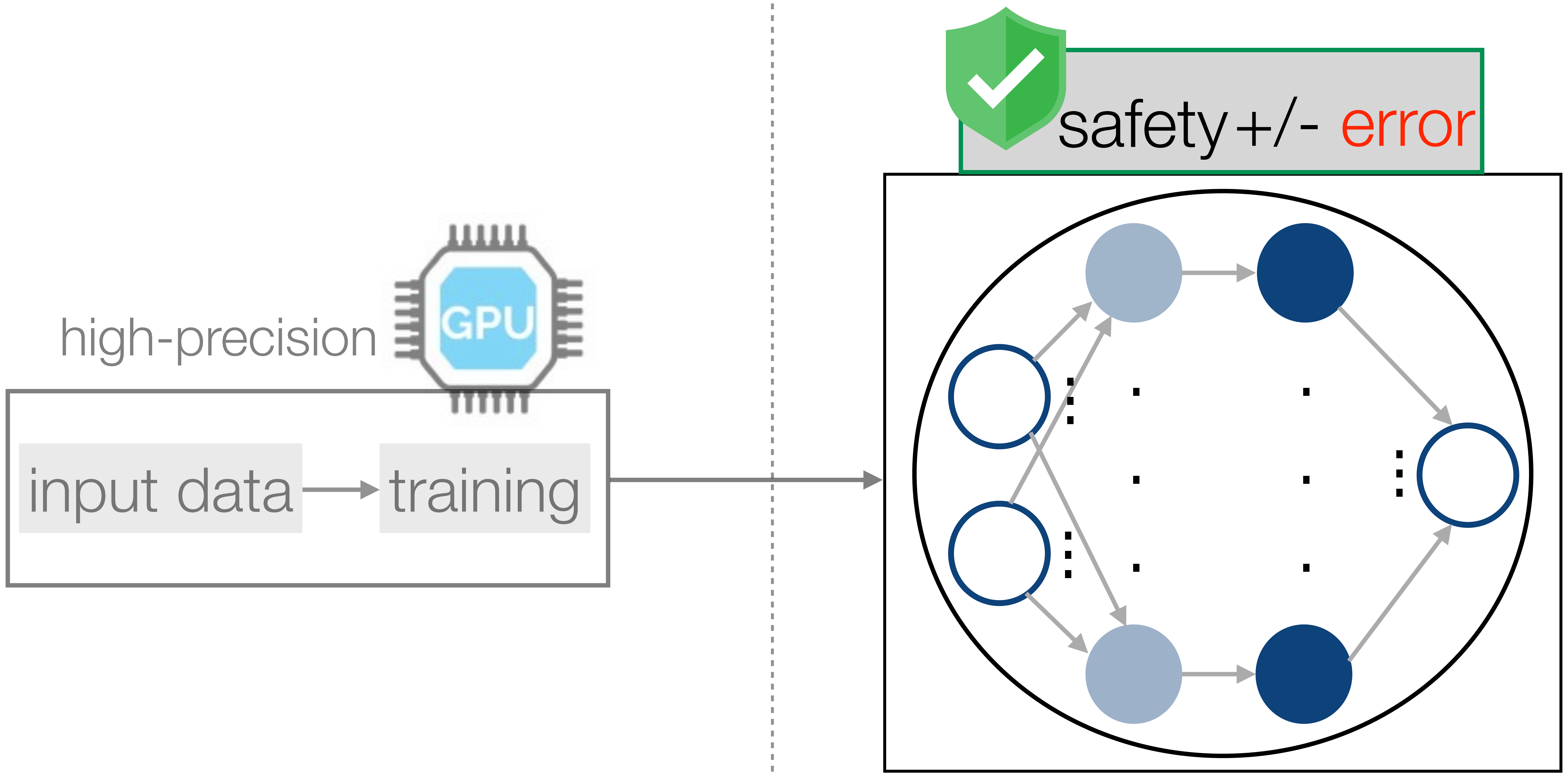




- ReachNN ’19
- Sherlock ’19
- NNV ’20
- …

real-valued arithmetic

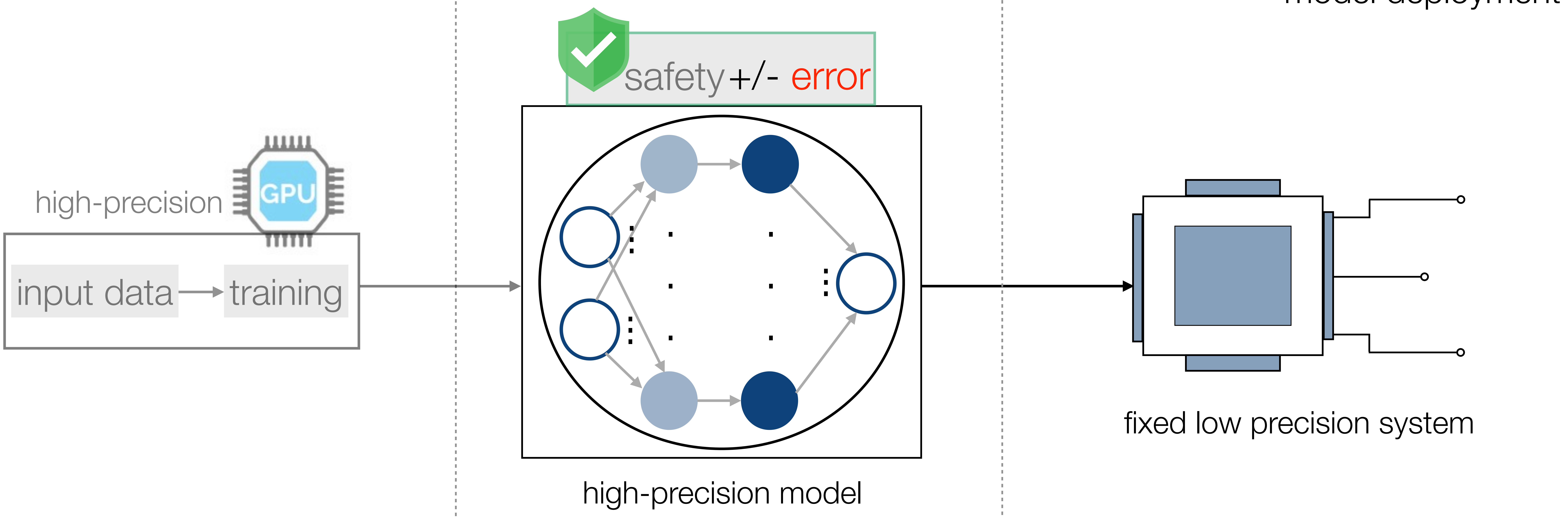
Safety Verification of Neural Network Controllers



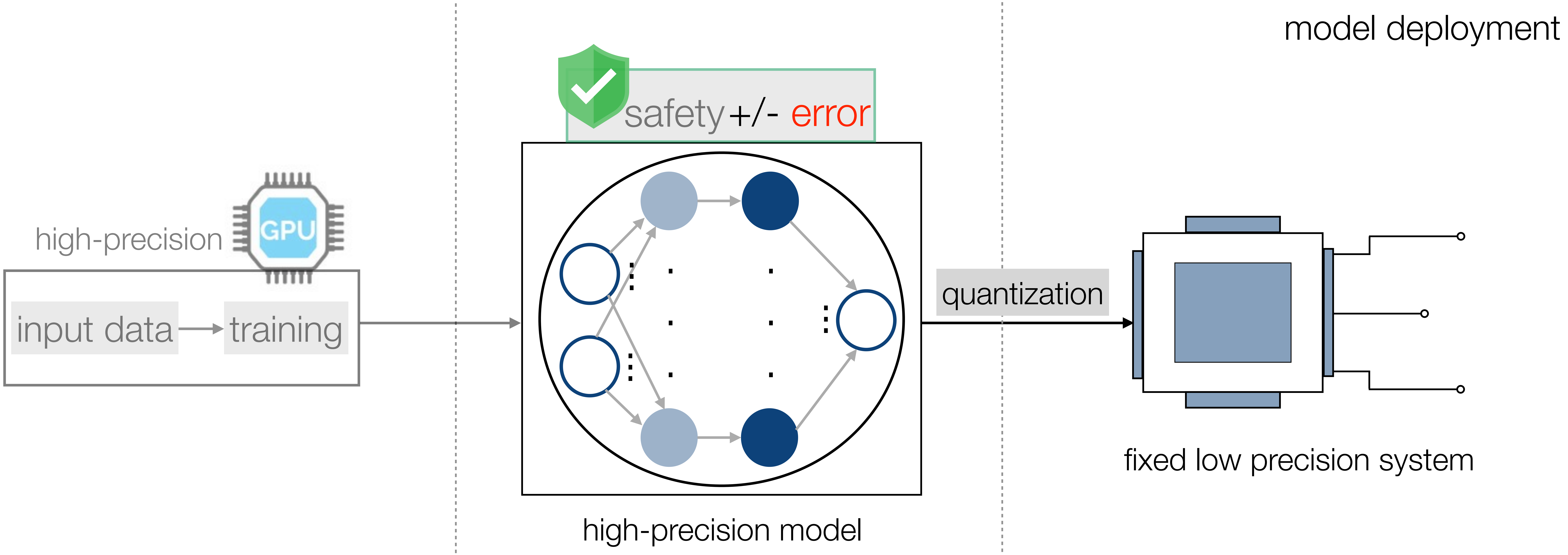
- ReachNN '19
- Sherlock '19
- NNV '20

Do not directly consider the finite-precision deployment of the model

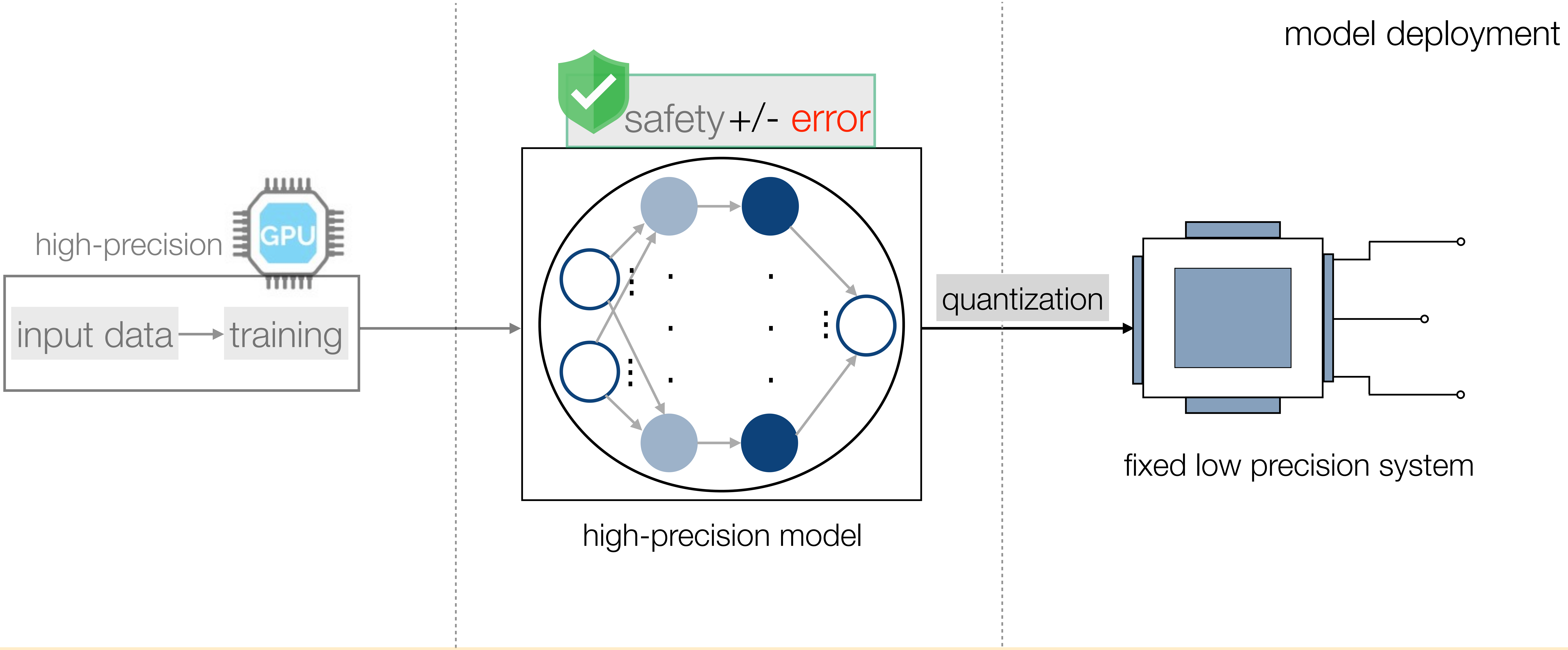
Controllers deployed in Embedded Systems



Controllers deployed in Embedded Systems



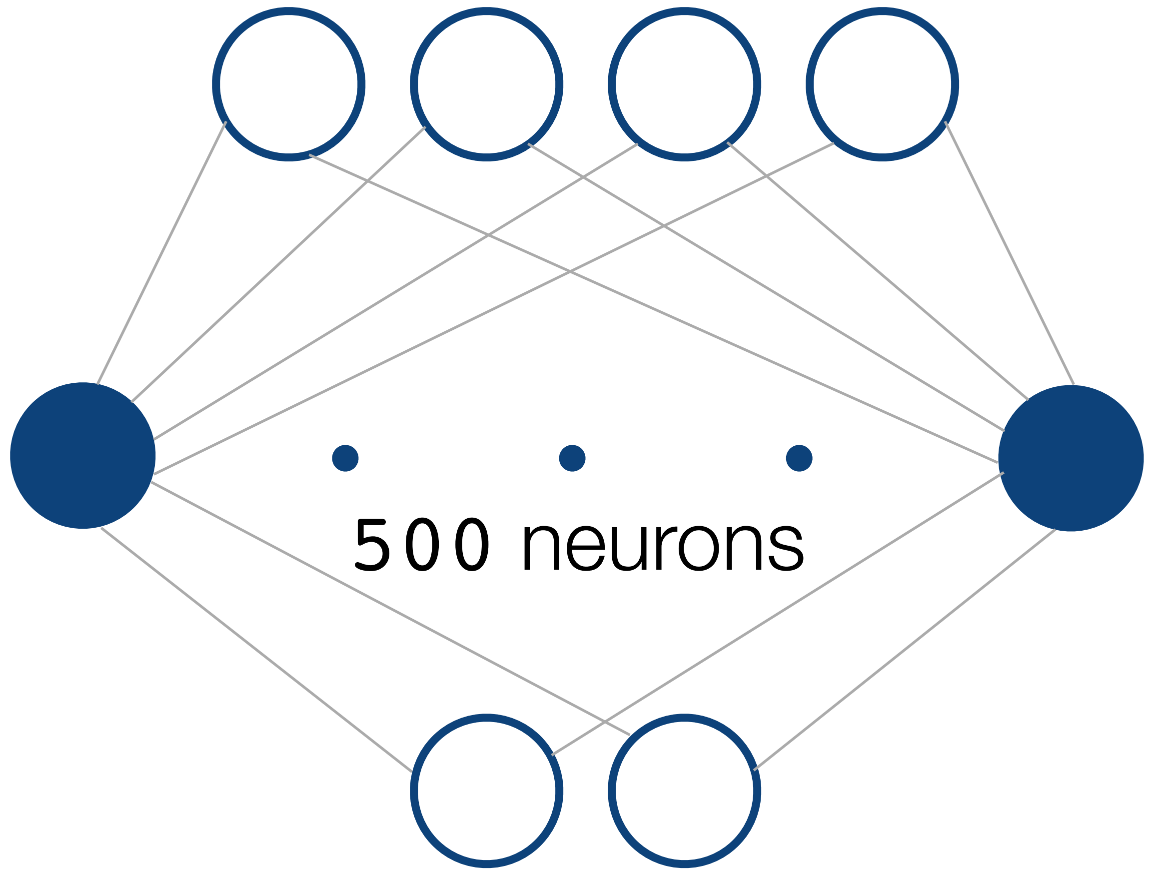
Neural Network Quantization



Our Goal: Quantize respecting the error bound!

Example: Unicycle Controller

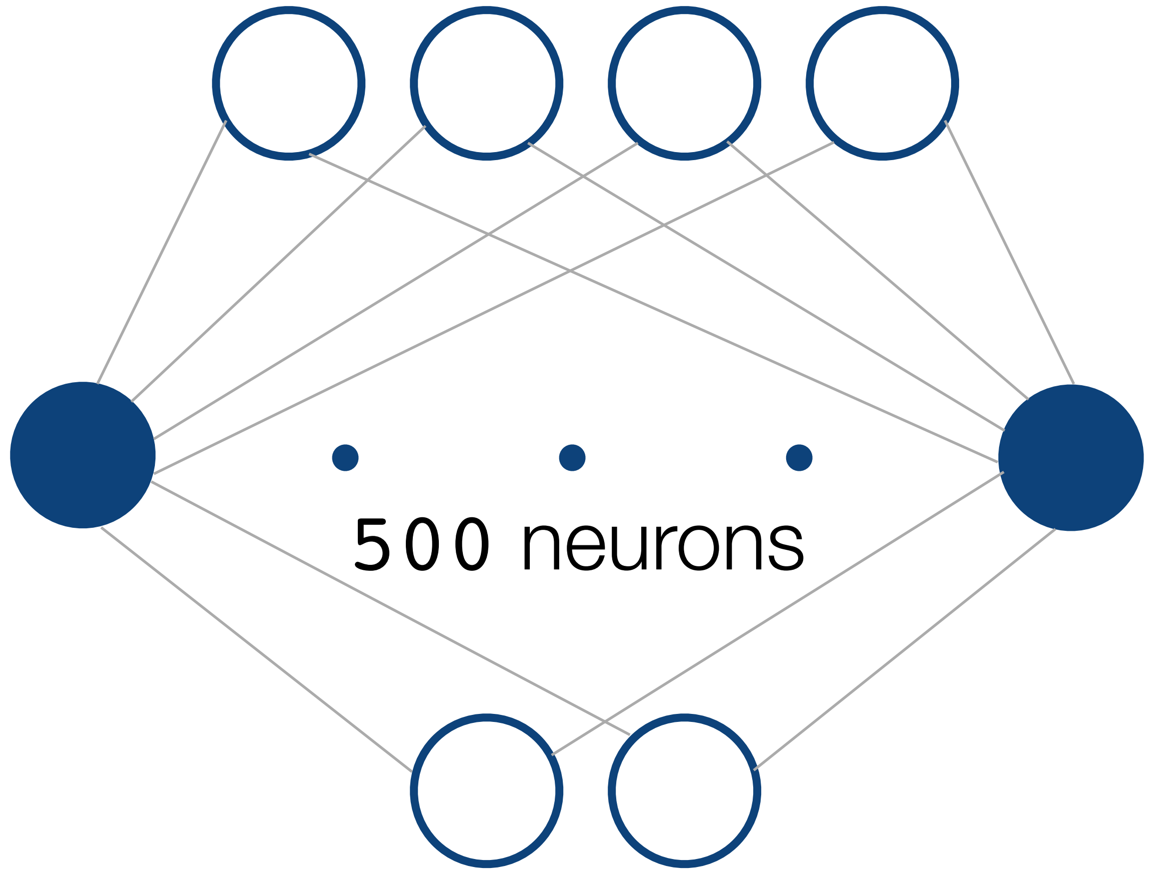
```
-0.6 <= in1 <= 9.55
-4.5 <= in2 <= 0.2
-0.06 <= in3 <= 2.11
-0.3 <= in4 <= 1.51
```




```
res +/- 1e-3
```

Example: Unicycle Controller

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




```
res +/- 1e-3
```

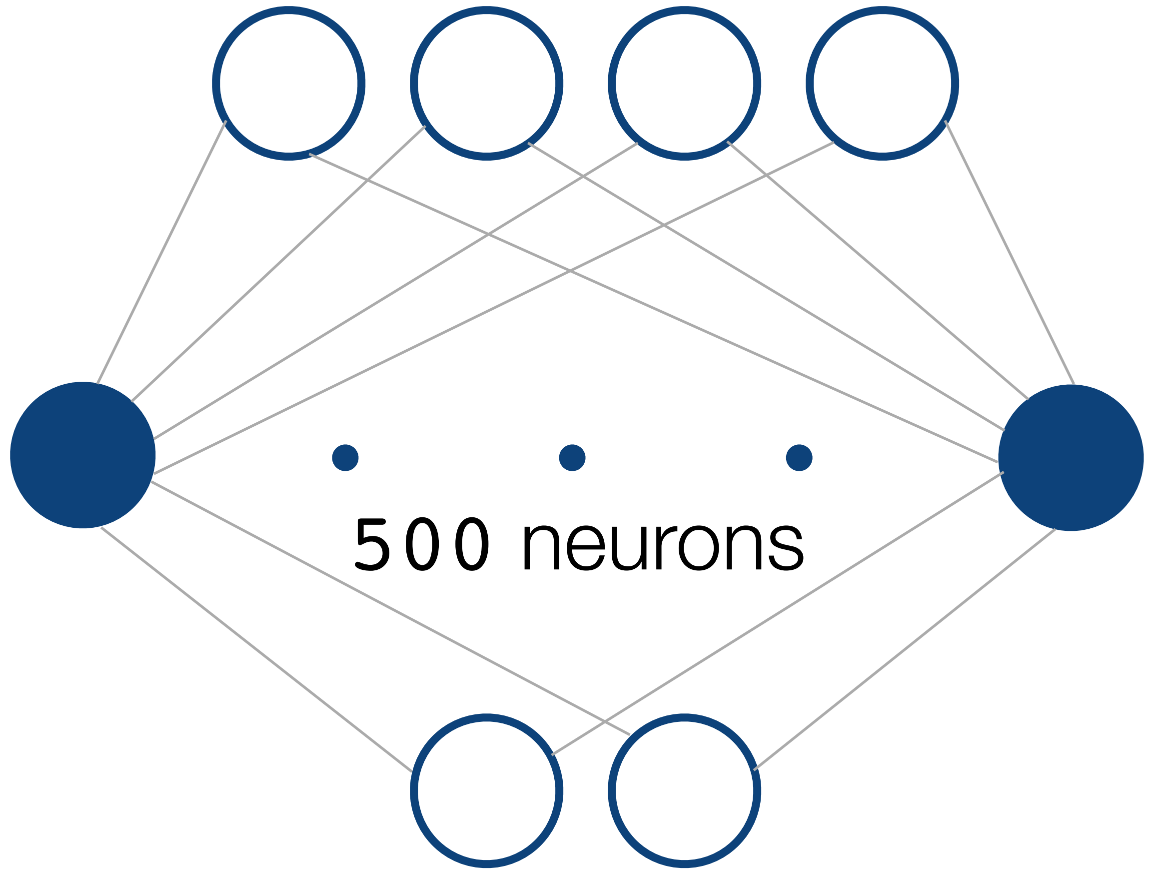
quantization

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<27,1>) (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
```




```
res +/- 1e-3
```

quantization

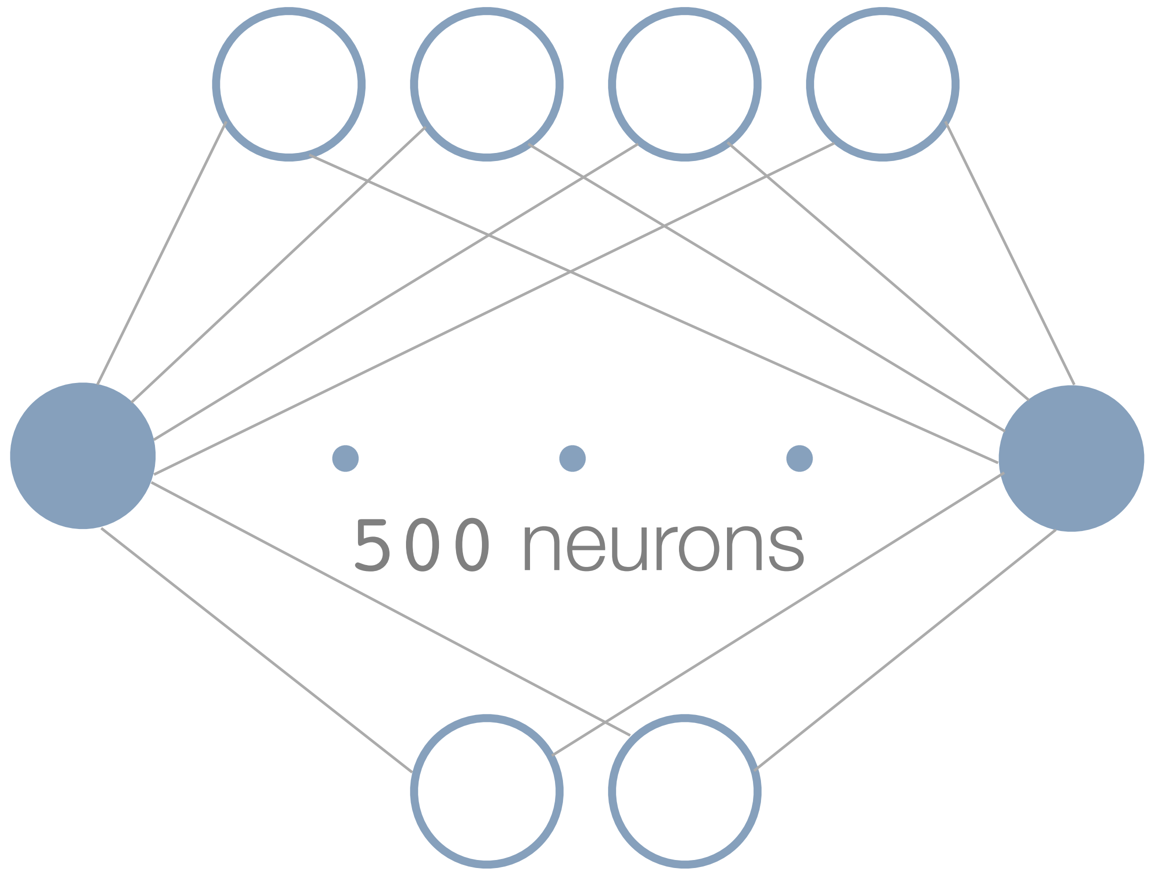
Aster

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<27,1>) (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
```




```
res +/- 1e-3
```

quantization

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<27,1>) (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;
}
```

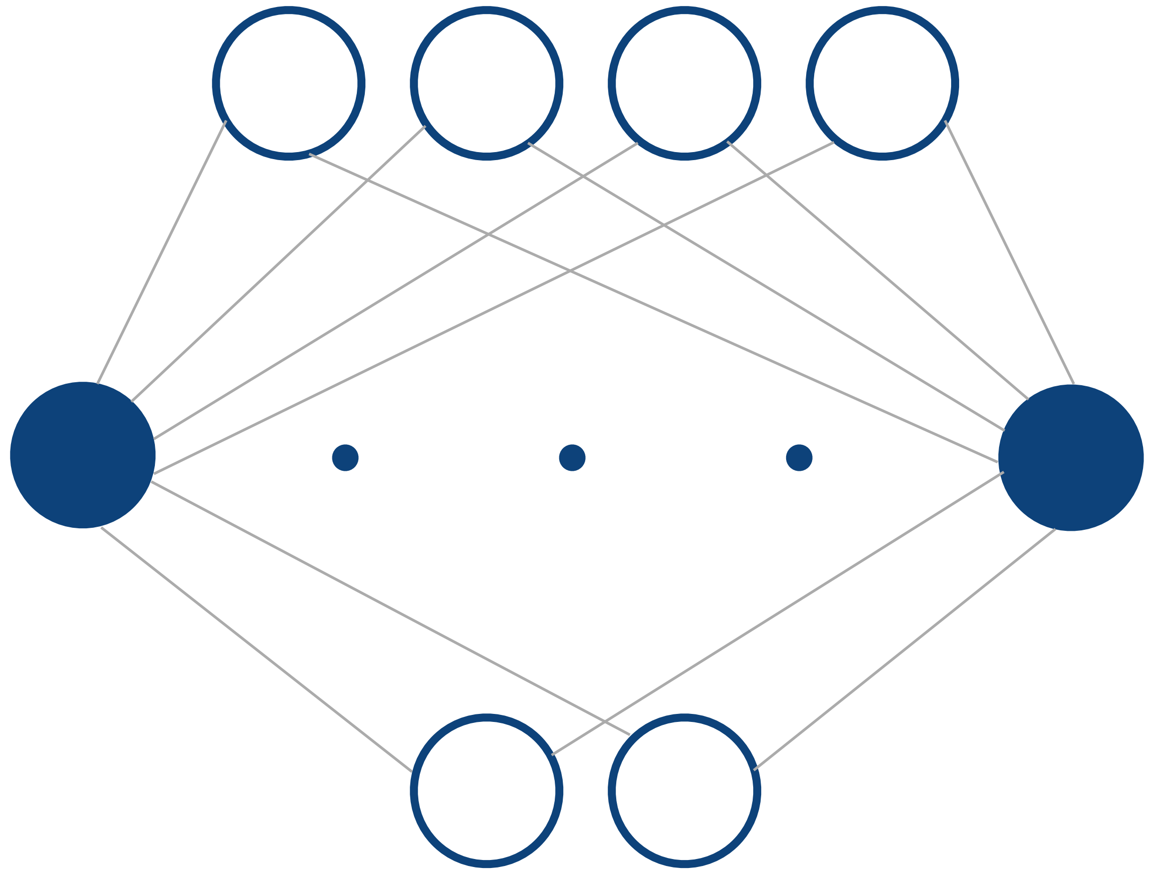
directly compiled

XILINX

latency = **27** cycles

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

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



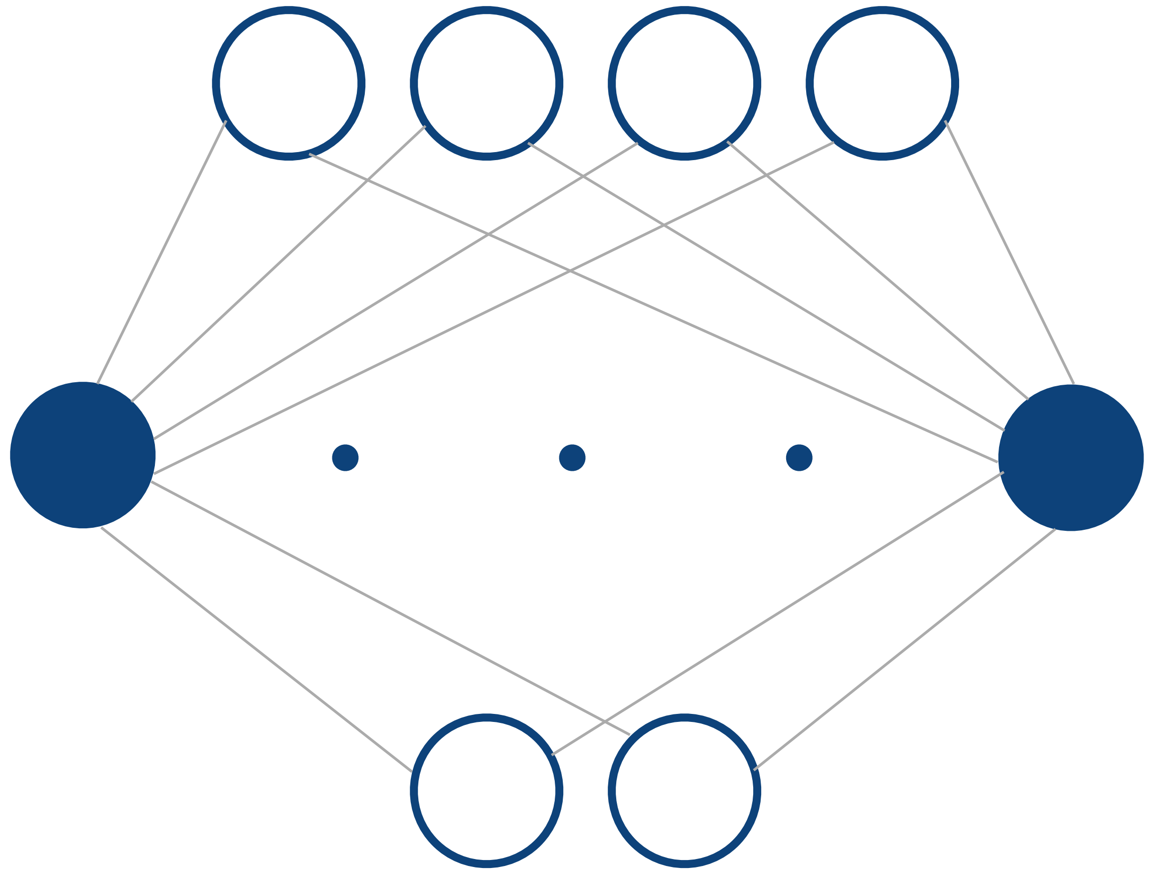
```
res +/- 1e-3
```

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



```
res +/- 1e-3
```

needs unrolled structures

Daisy[1]

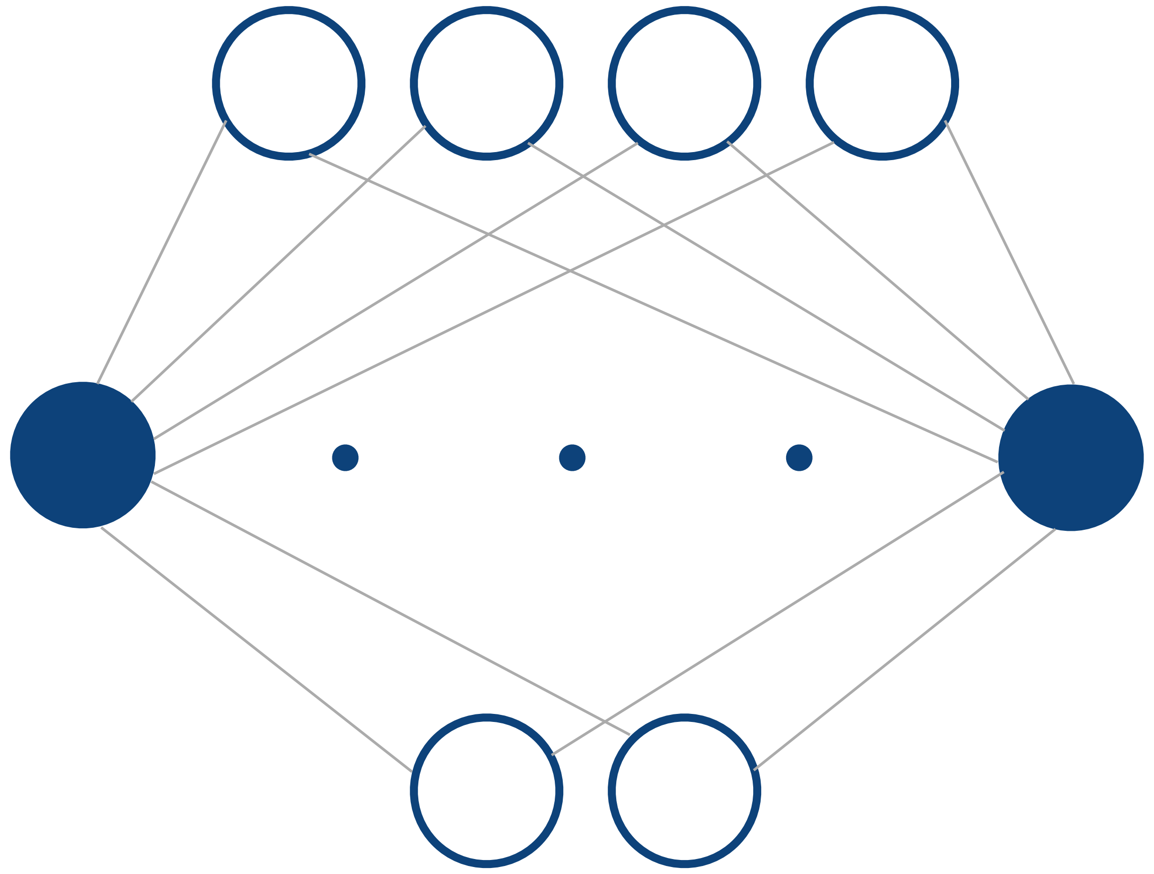
latency = **178** cycles

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



```
res +/- 1e-3
```

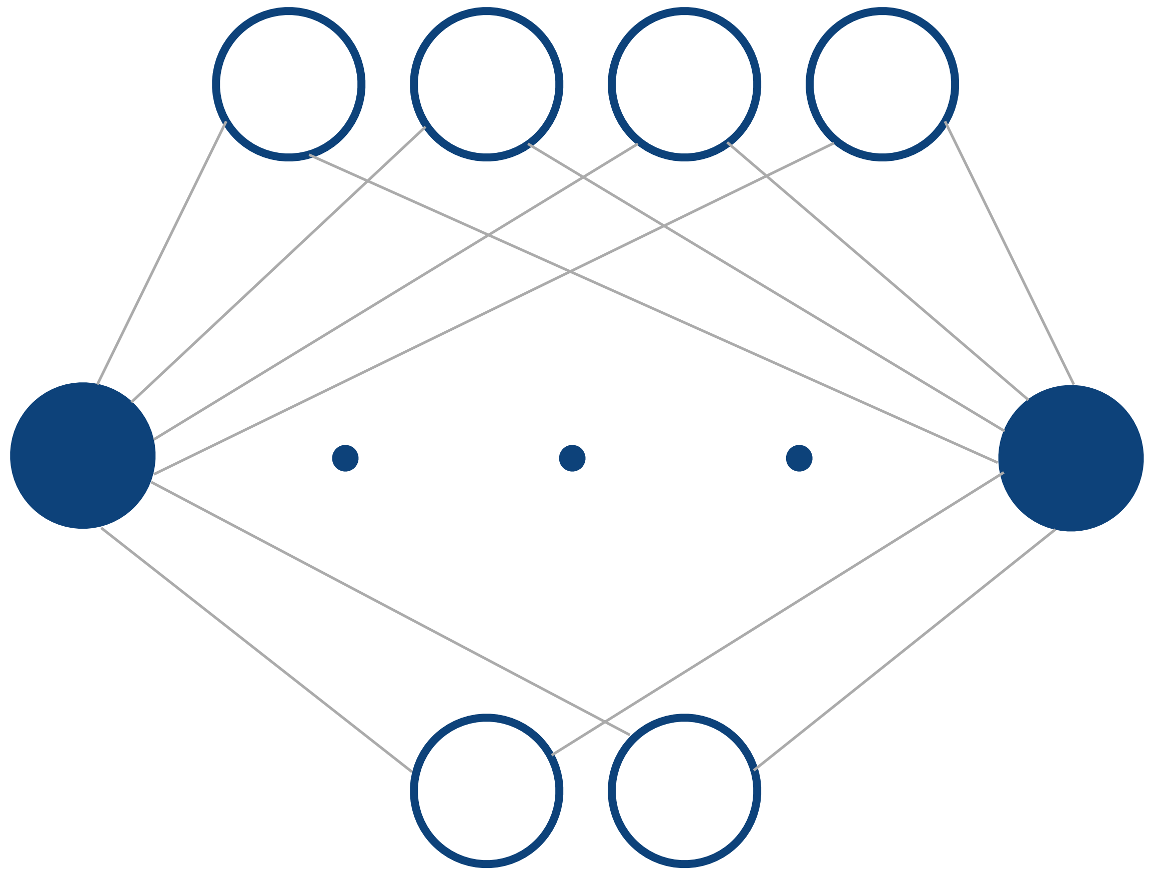
FPTuner[2]

Daisy[1]

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

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

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



```
res +/- 1e-3
```

FPTuner[2]

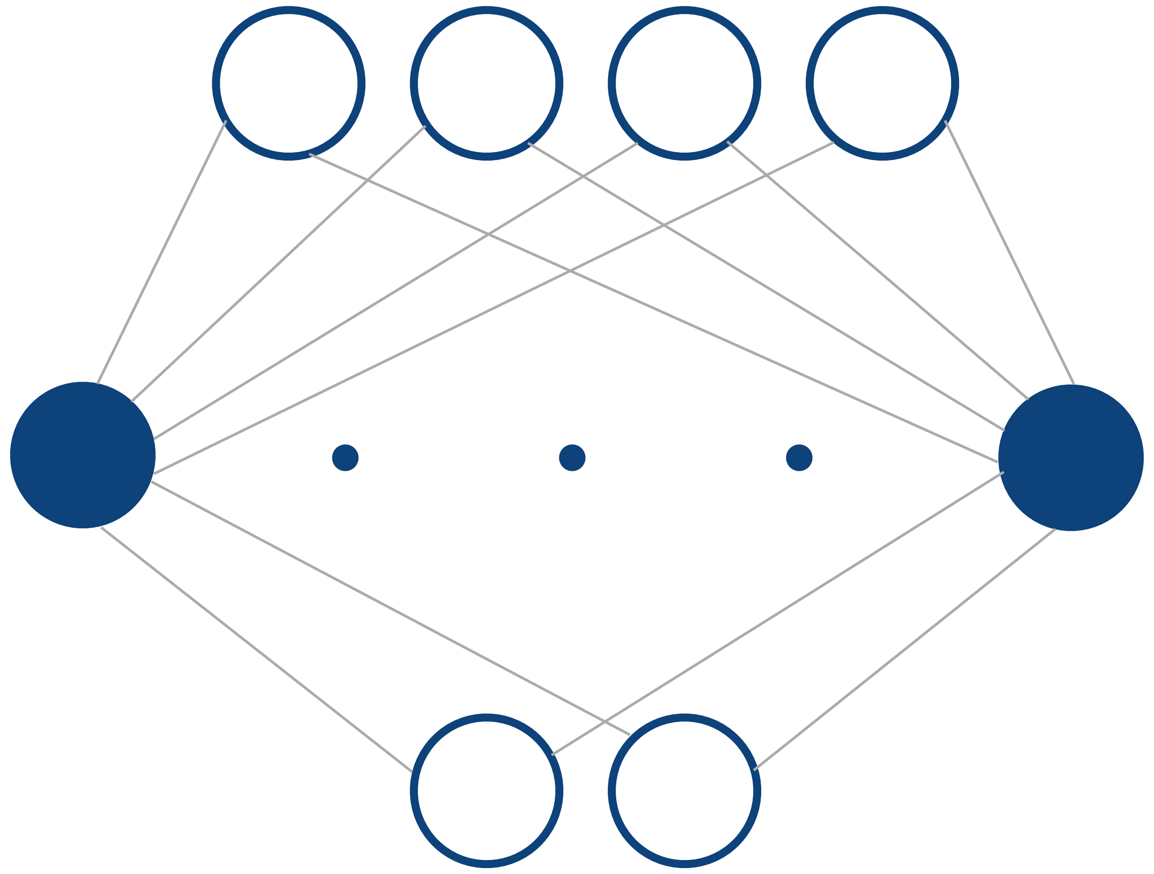
no fixed-point support!

Daisy[1]

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

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
```

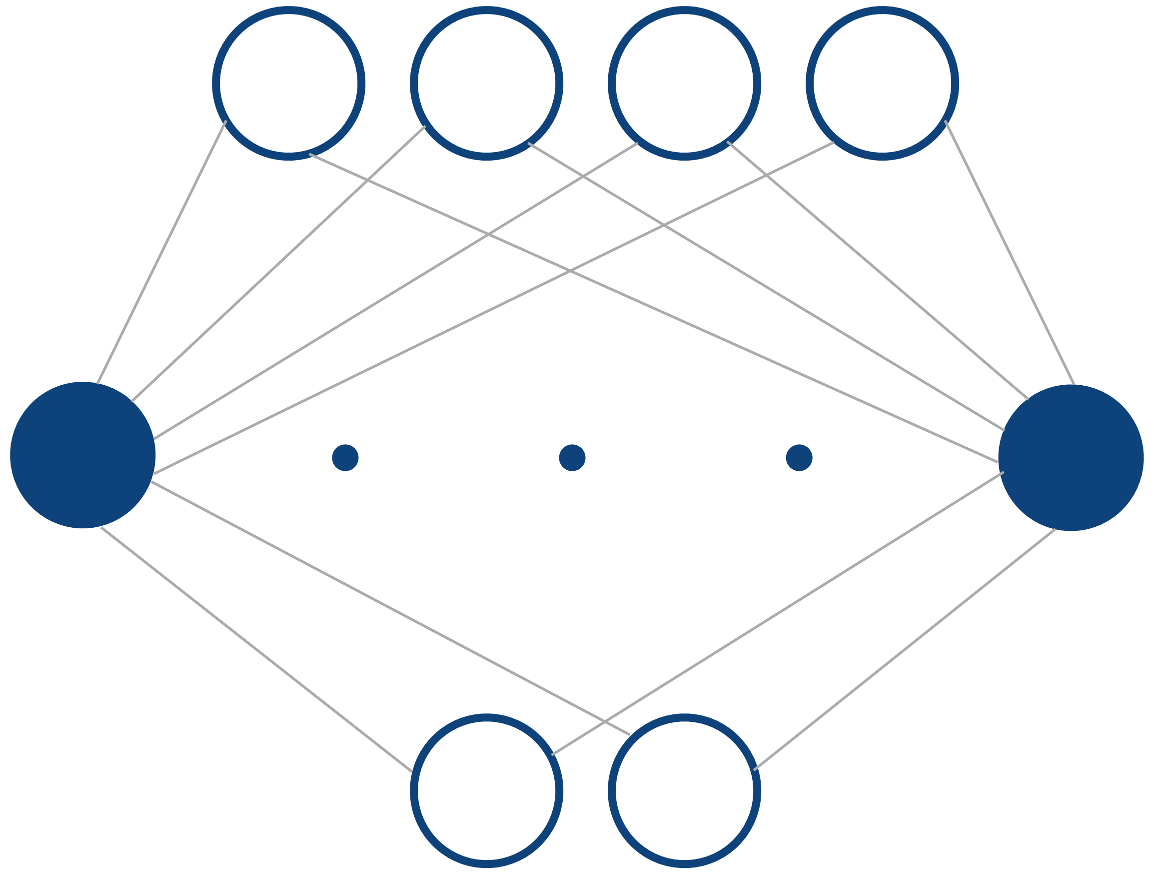
...
Shiftry[2]
SeeDot[1]

FPTuner[2]
Daisy[1]

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]

does not work for controllers!

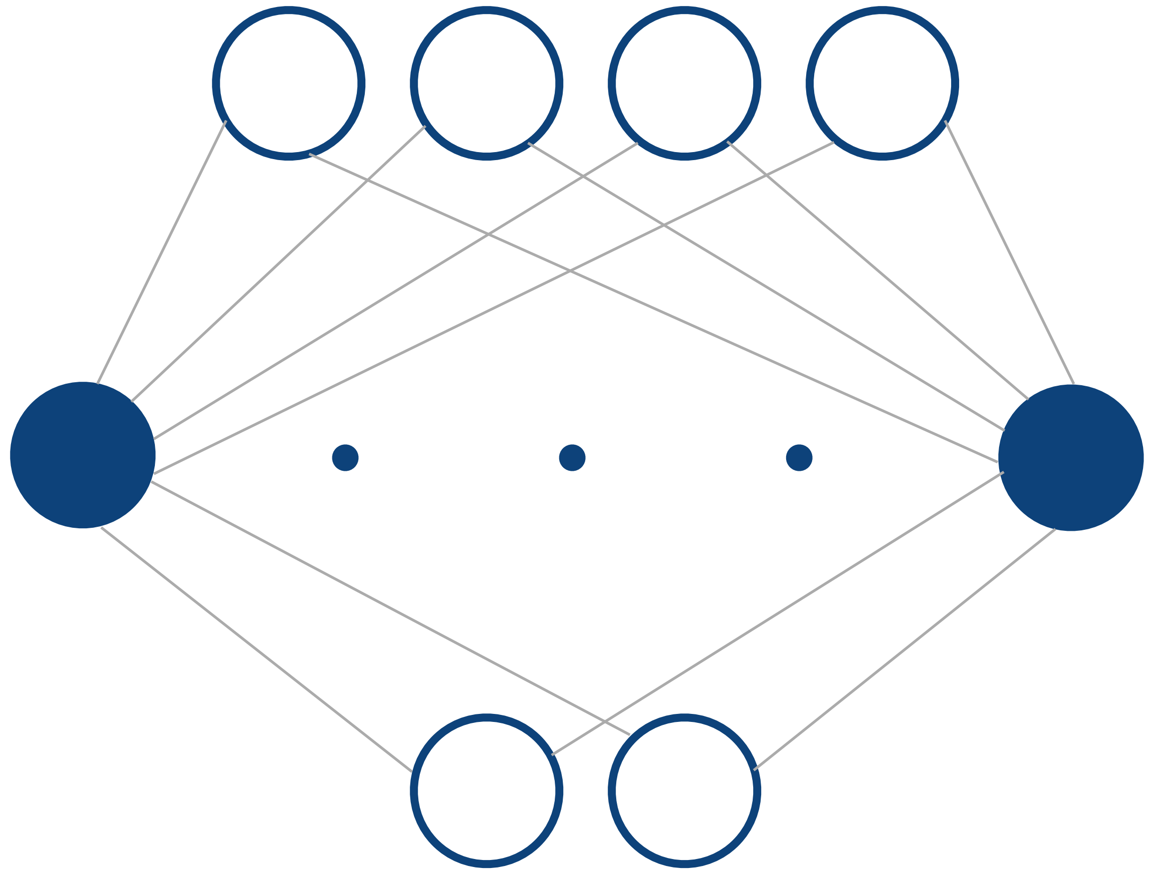
FPTuner[2]

Daisy[1]

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
```

unsound!

...

Shiftry[2]

SeeDot[1]

does not work for controllers!

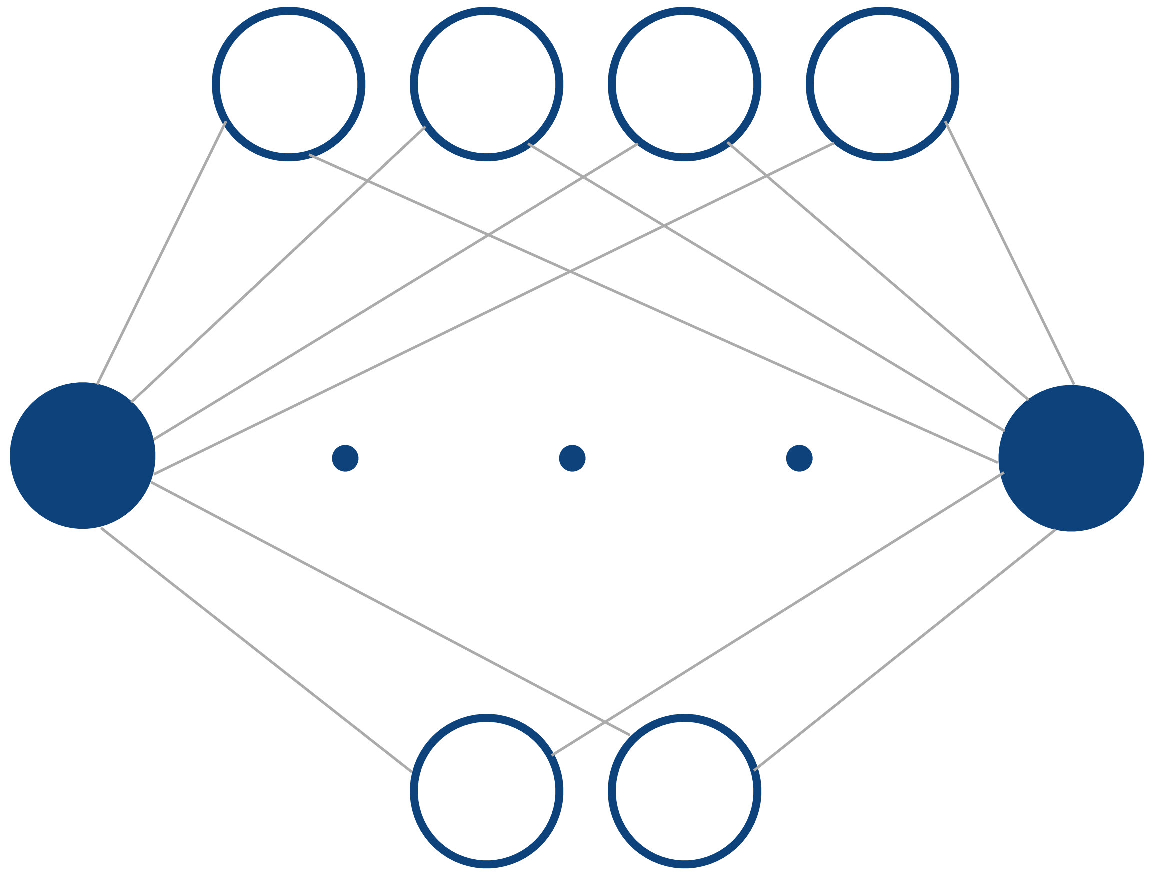
FPTuner[2]

Daisy[1]

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 <= 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]

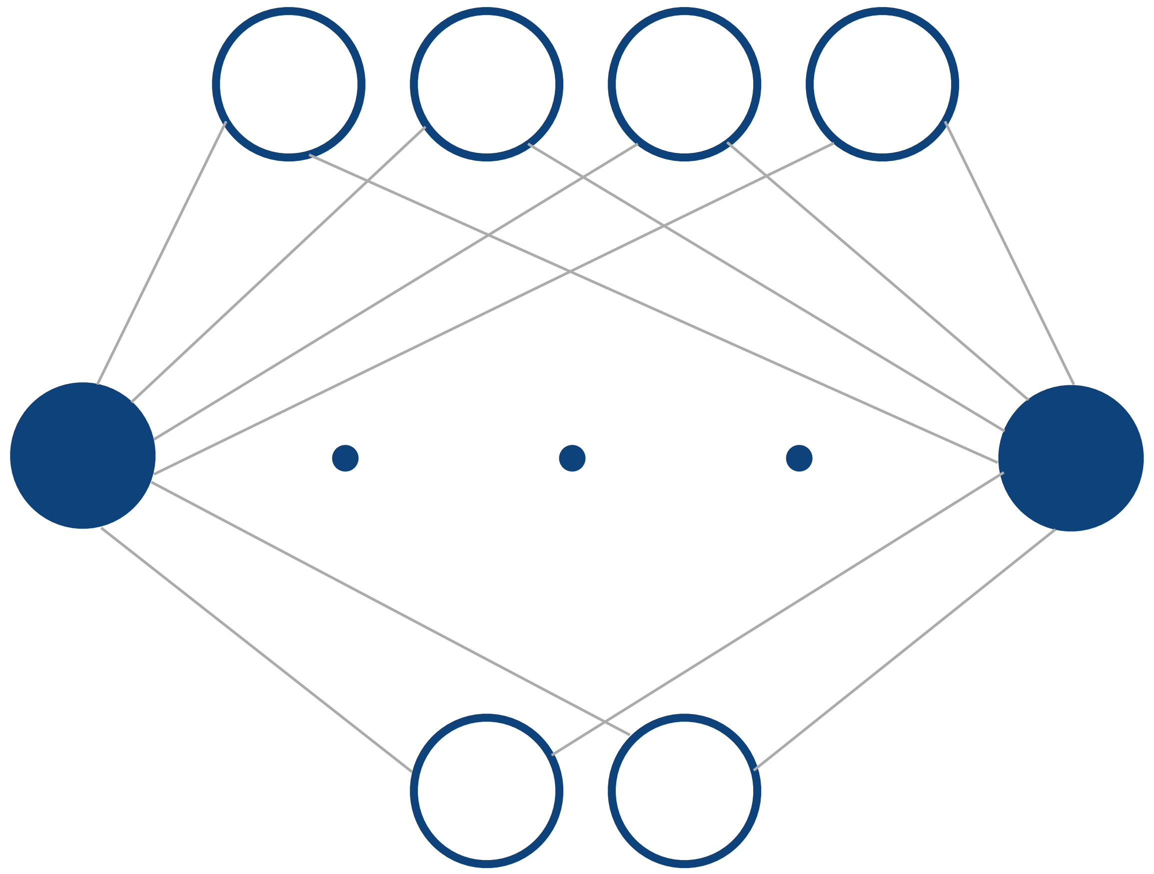
unsound quantizers
for classifiers

FPTuner[2]
Daisy[1]

sound tuner for
numerical programs

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
```

...
Shiftry[2]
SeeDot[1]

unsound quantizers for classifiers

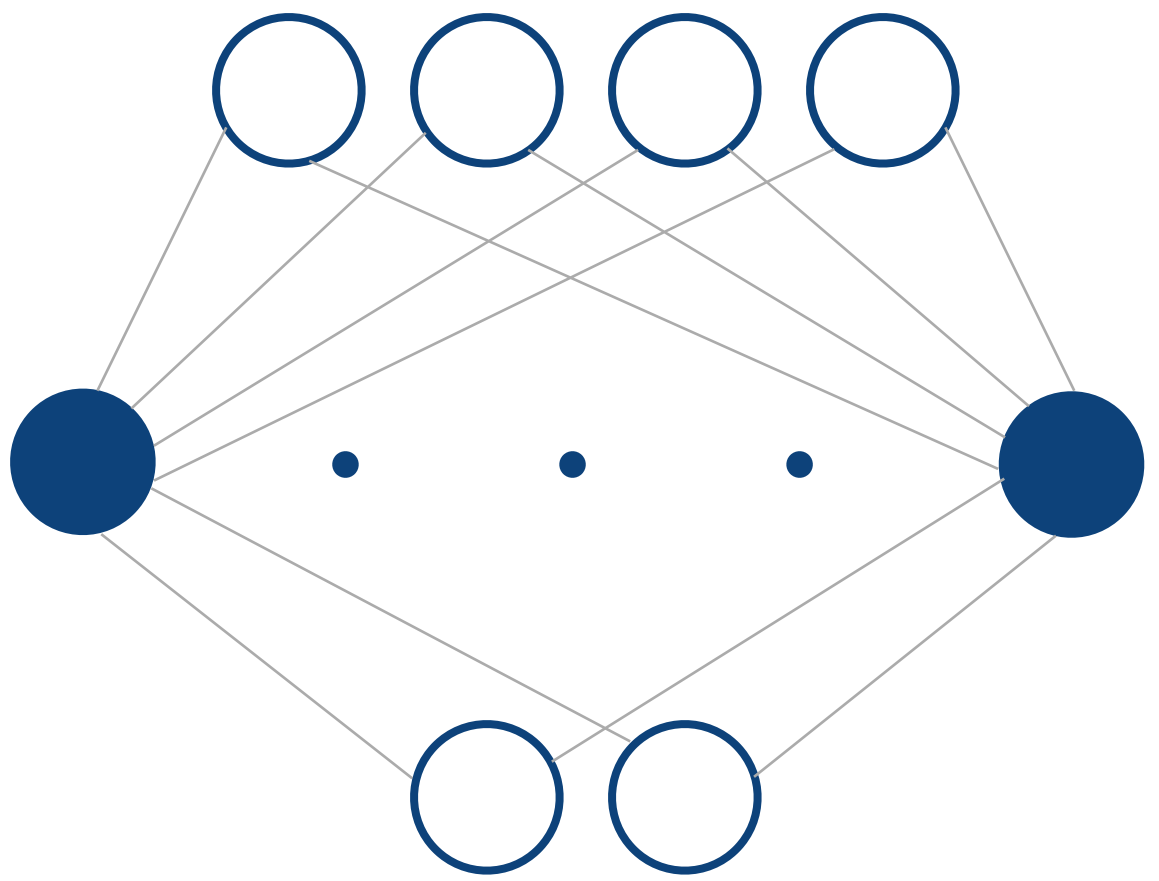
FPTuner[2]
Daisy[1]

sound tuner for numerical programs

We provide: sound quantizer for NN controllers guaranteeing error bounds

Key Idea: Quantization for efficiency is an optimization problem!

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

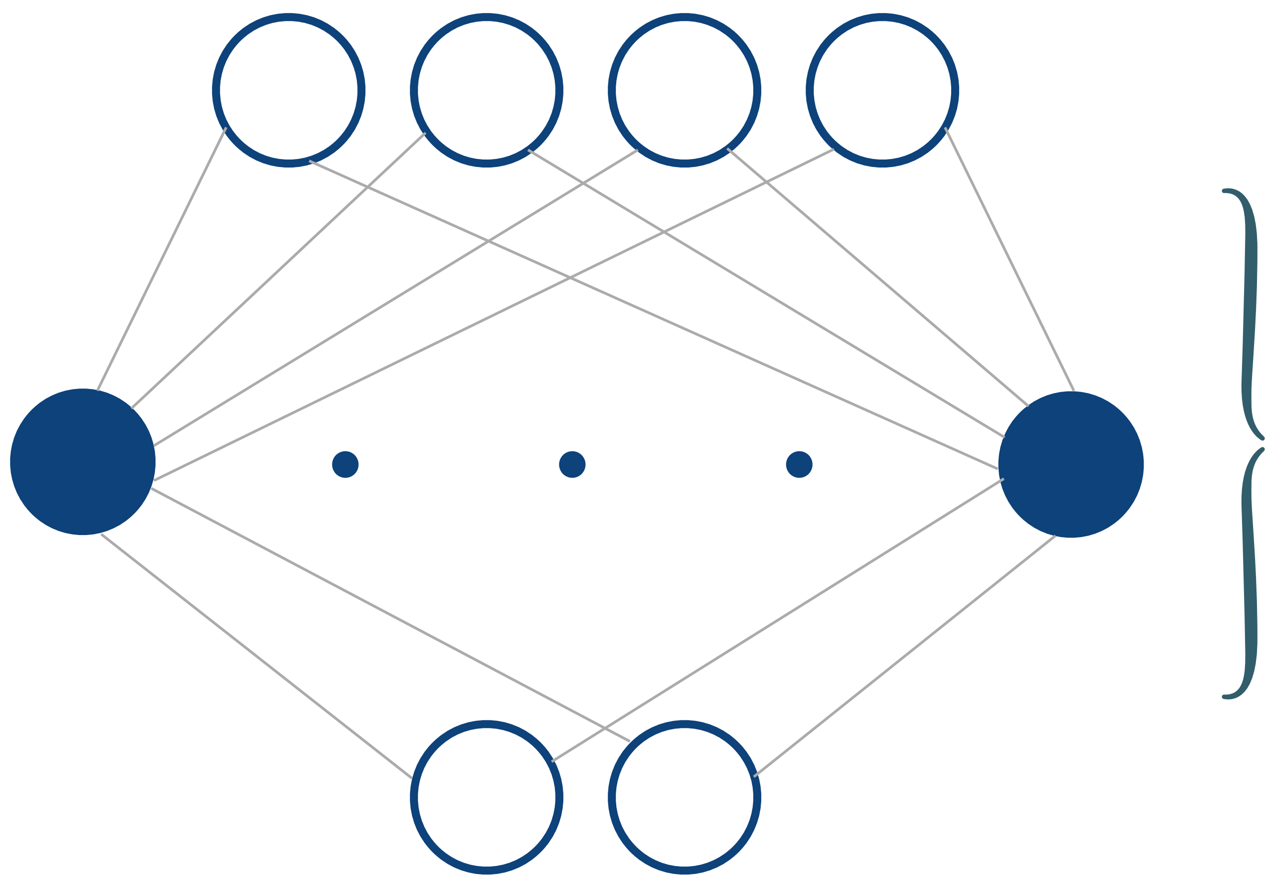


```
minimize: precision
```

```
res +/- 1e-3
```

Key Idea: Quantization for efficiency is an optimization problem!

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



```
minimize: precision
```

cost function

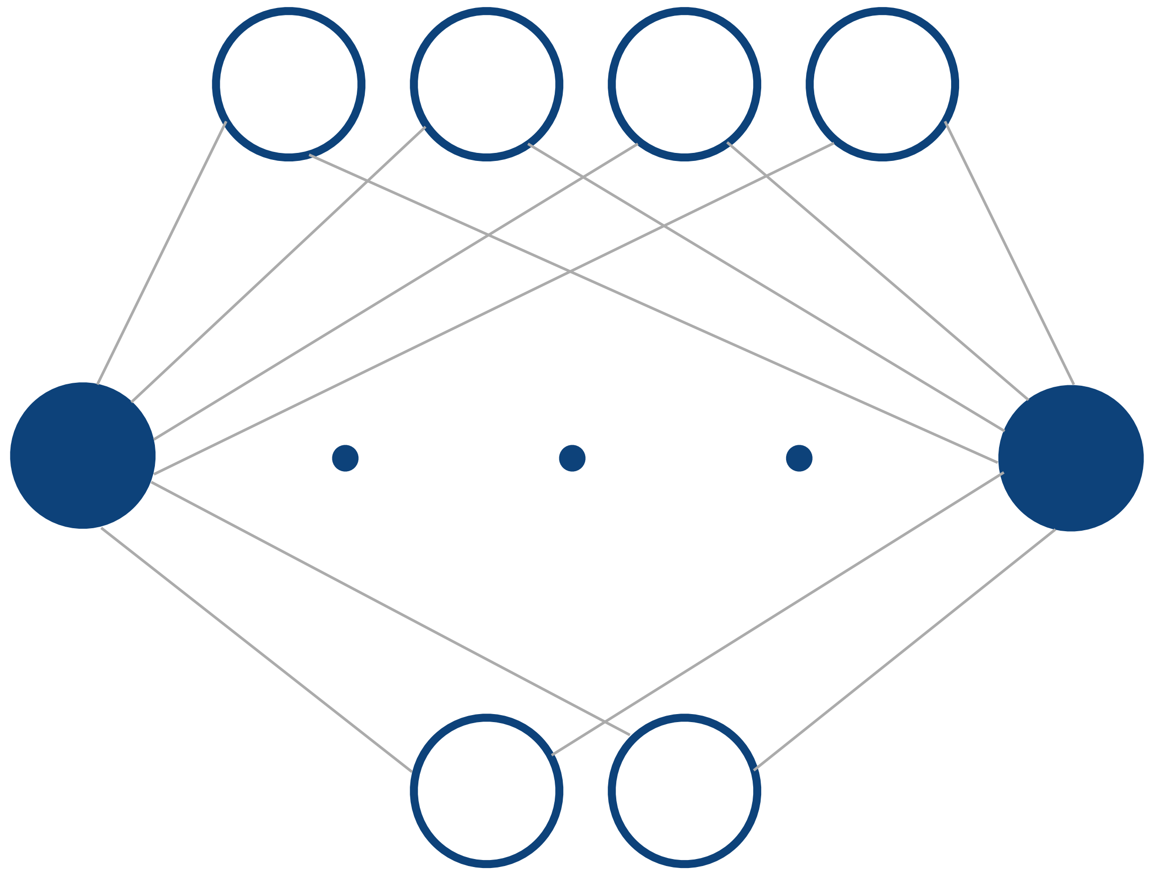
```
res +/- 1e-3
```

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

- integer-valued cost

Key Idea: Quantization for efficiency is an optimization problem!

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



```
res +/- 1e-3
```

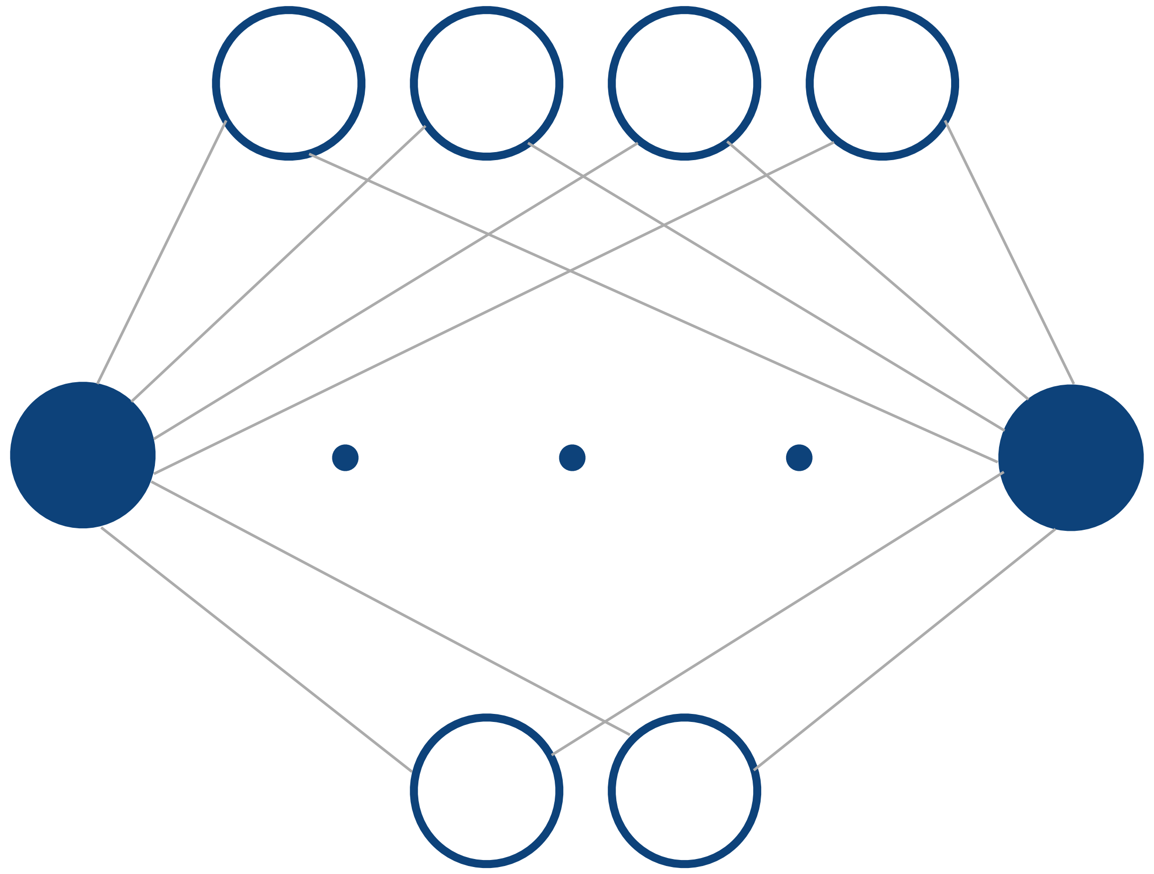
error constraint

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

- integer-valued cost

Key Idea: Quantization for efficiency is an optimization problem!

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



```
res +/- 1e-3
```

error constraint

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

subject to:

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

- integer-valued cost
- real-valued error constraint

Quantization Formulated as an Optimization Problem

mixed-integer problem

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

subject to:

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

- integer-valued cost
- real-valued error constraint

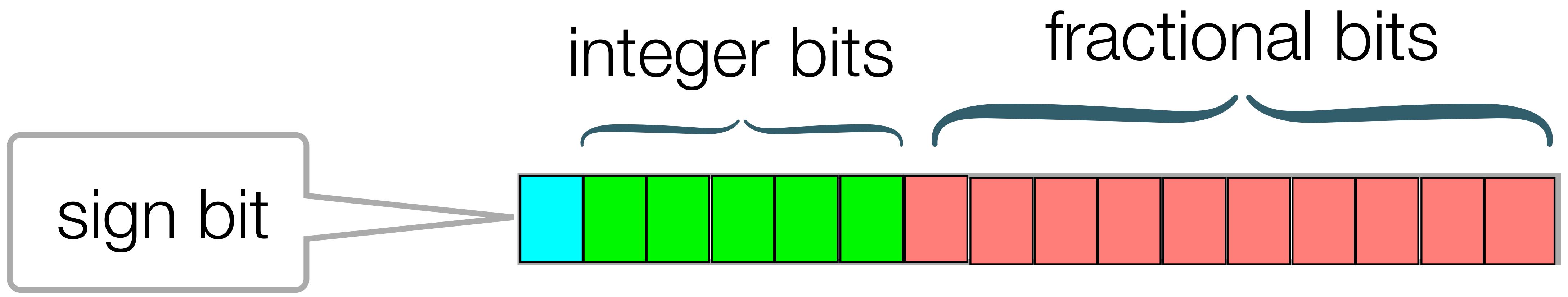
Quantization in Fixed-Point Precision

mixed-integer problem

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

subject to:

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



fixed-point representation

Quantization in Fixed-Point Precision

mixed-integer problem

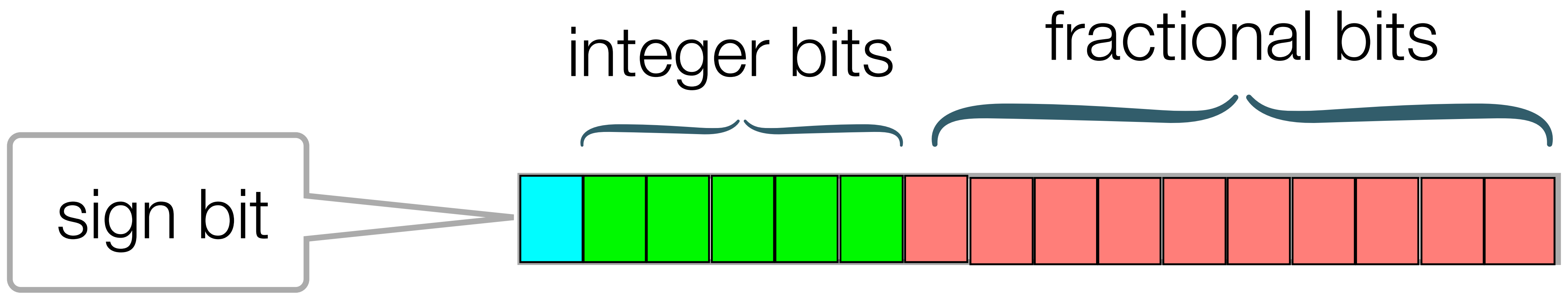
minimize: $\gamma = \sum_{i=1}^{n} \gamma_i^{dot} + \gamma_i^{bias} + \gamma_i^{\alpha}$

subject to:

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

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

constraint ensuring no overflow



fixed-point representation

Quantization in Fixed-Point Precision

mixed-integer problem

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

subject to:

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

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

mixed-integer non-linear hard problem!

Quantization in Fixed-Point Precision

mixed-integer problem

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

subject to:

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

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

mixed-integer non-linear hard problem!

Quantization in Fixed-Point Precision

mixed-integer problem

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

subject to:

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

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

mixed-integer non-linear hard problem!

Our Idea: Reduce to Mixed Integer <u>Linear</u> Programming (MILP) Problem!

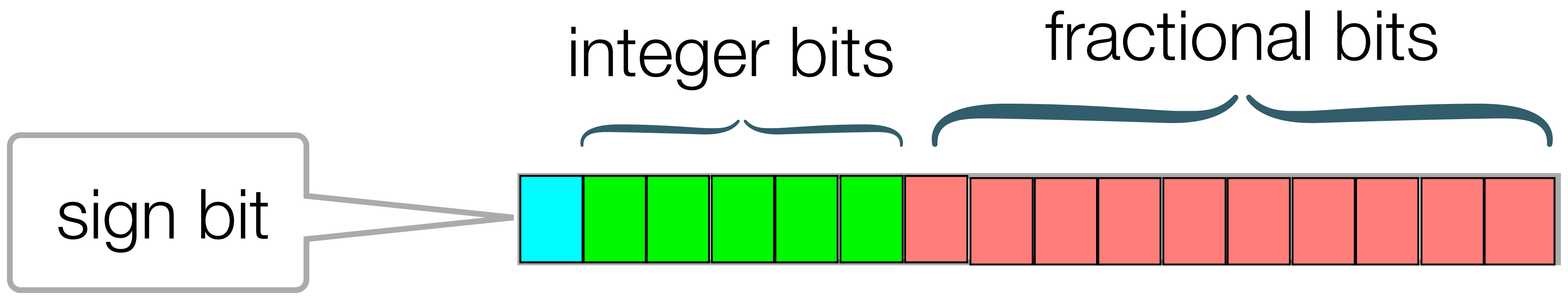
Overview: Reduction to MILP

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

subject to:

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

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



fixed-point representation

Overview: Reduction to MILP

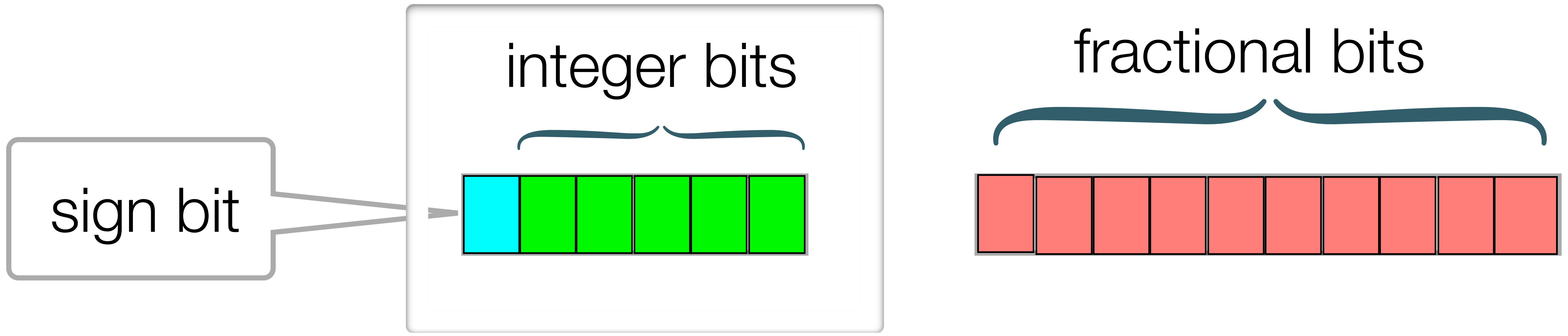
minimize: $\gamma = \sum_{i=1}^{n} \gamma_i^{dot} + \gamma_i^{bias} + \gamma_i^{\alpha}$

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



fixed-point representation

Overview: Reduction to MILP

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

subject to:

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

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

linearize exactly

- over-approximate integer bits separately

Overview: Reduction to MILP

abstract dot product

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

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

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

Overview: Reduction to MILP

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

subject to:

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

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

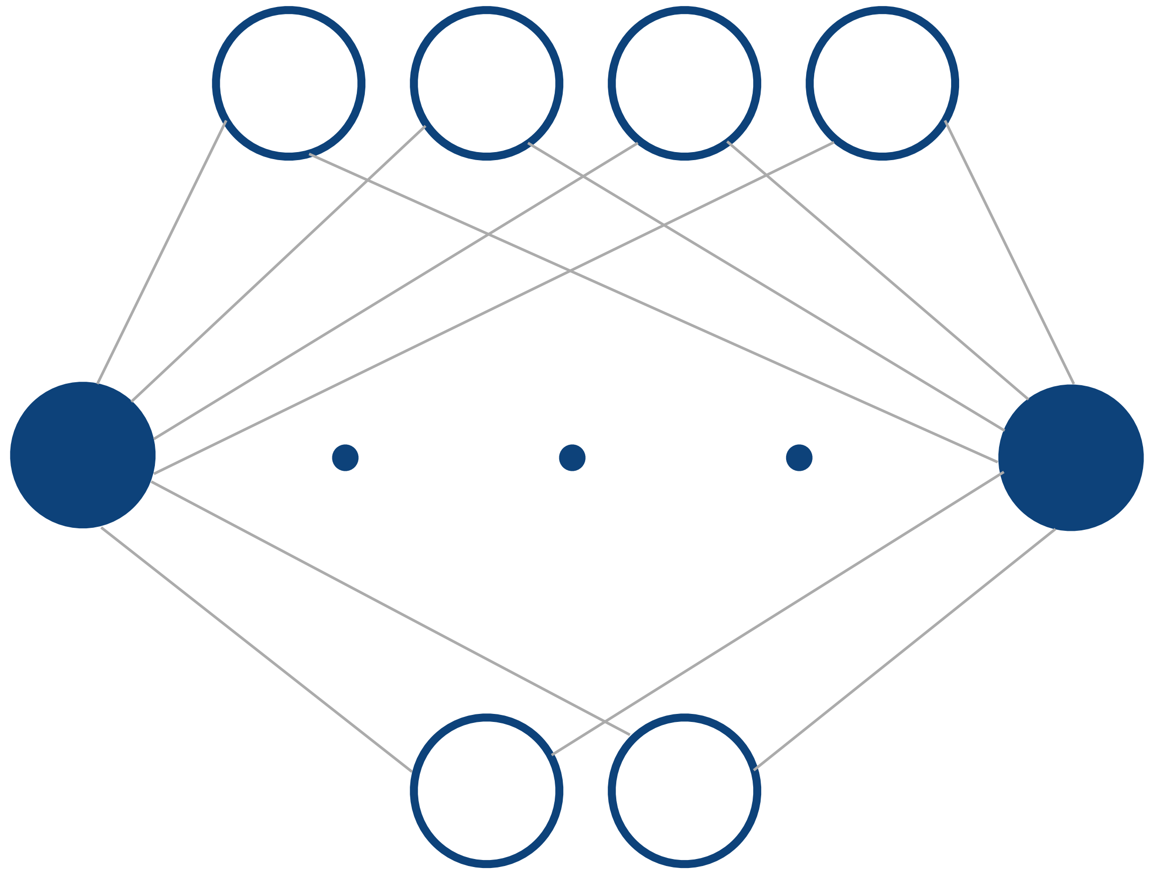
- over-approximate integer bits separately
- linearize bias cost and error constraint exactly
- abstract dot product

Linearization Step 1: Computing Integer Bits

step 1: computing **integer bits** using interval arithmetic

Linearization Step 1: Computing Integer Bits

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



`res +/- 1e-3`

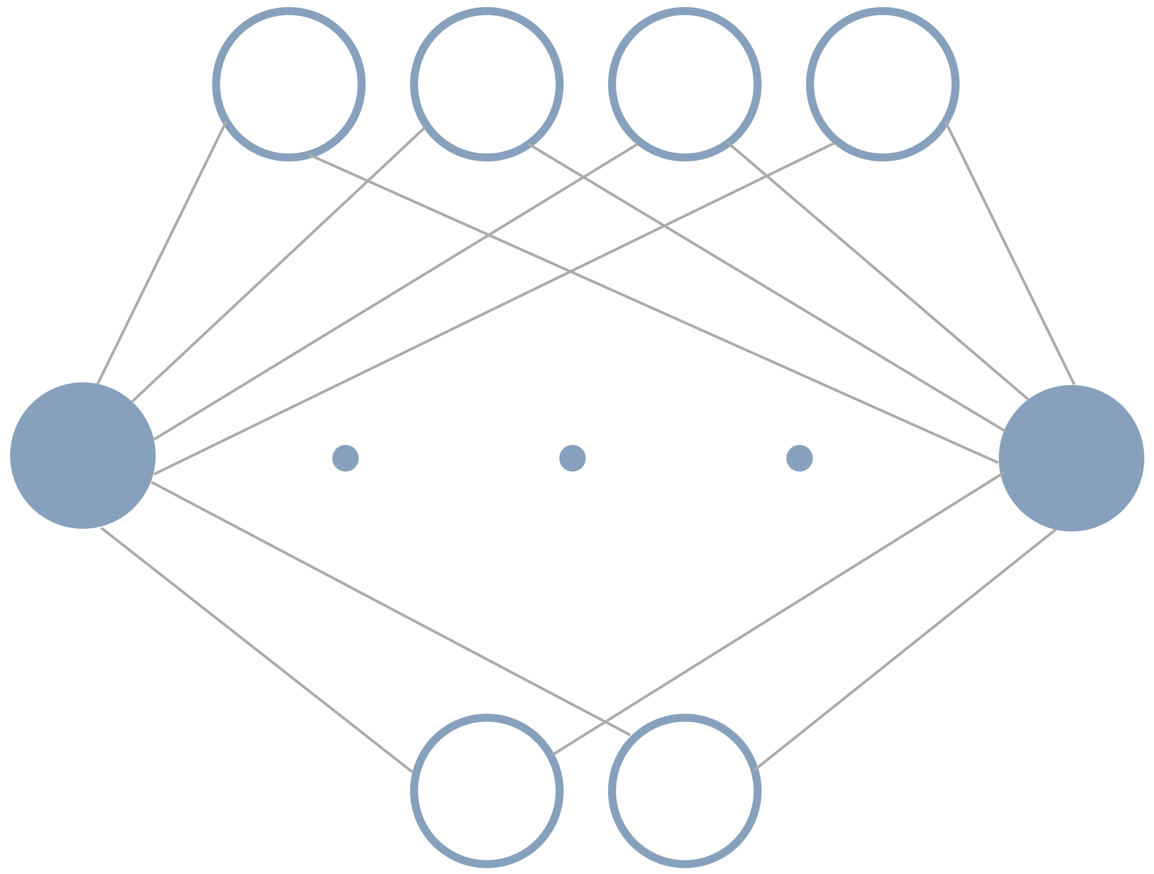
$$\begin{bmatrix}[-0.6,9.55]\\ [-4.5,0.2]\\ [-0.06,2.11]\\ [-0.3,1.51]\end{bmatrix} \quad \texttt{in} \qquad \texttt{weights1} \quad \begin{bmatrix}[-0.034,-0.01,-0.13,0.04]\\ \dots\\ [0.13,0.1,0.05,-0.17]\end{bmatrix}$$

$$* \quad \begin{bmatrix}[\,]\\ \dots\\ [\,]\end{bmatrix}$$

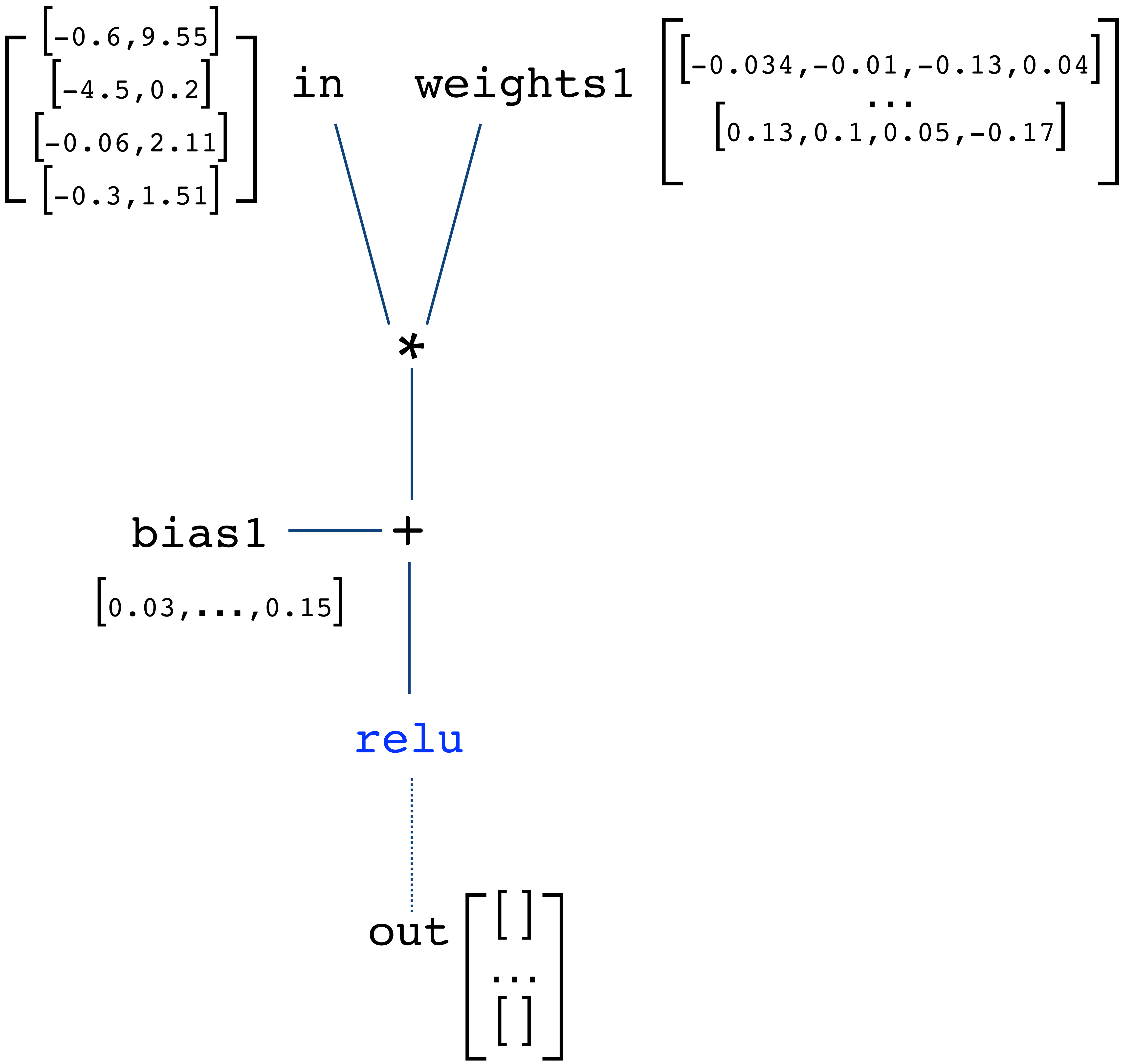
compute integer bits for variables

Linearization Step 1: Computing Integer Bits

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

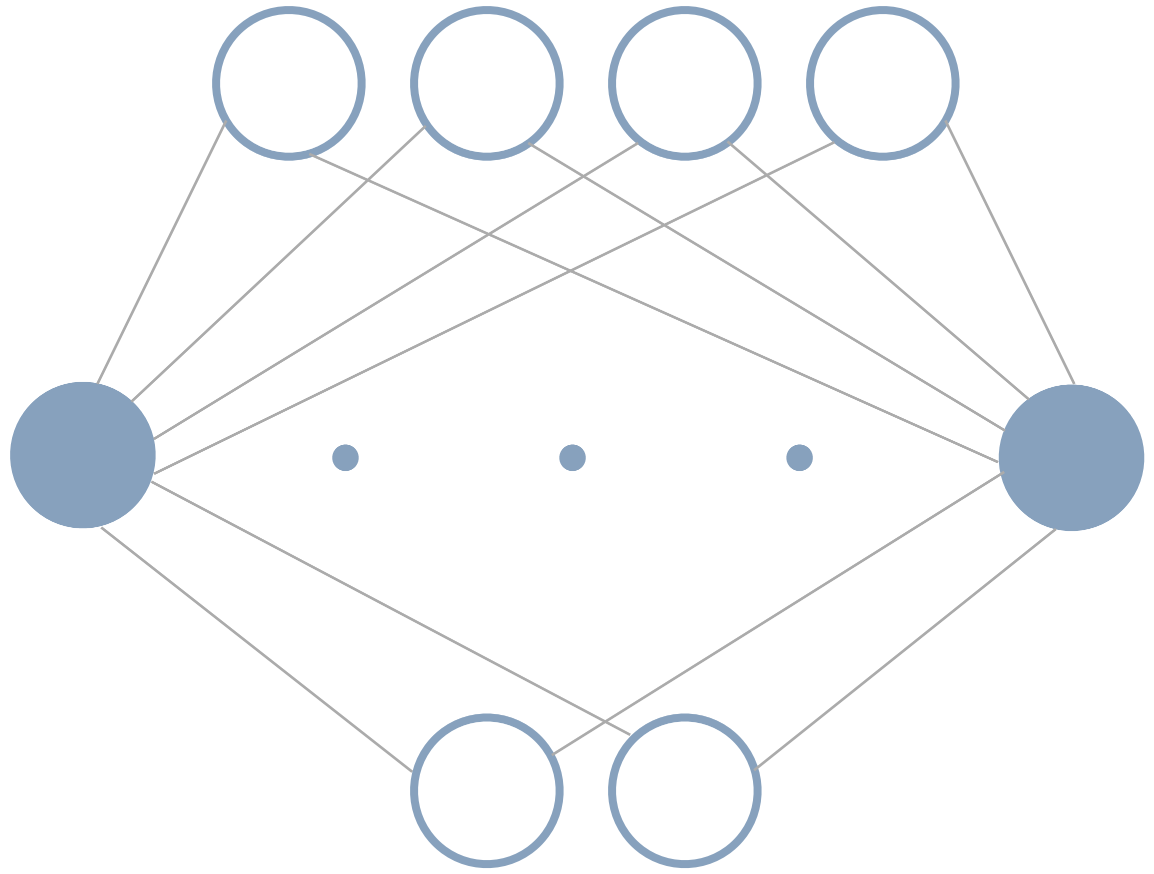


```
res +/- 1e-3
```

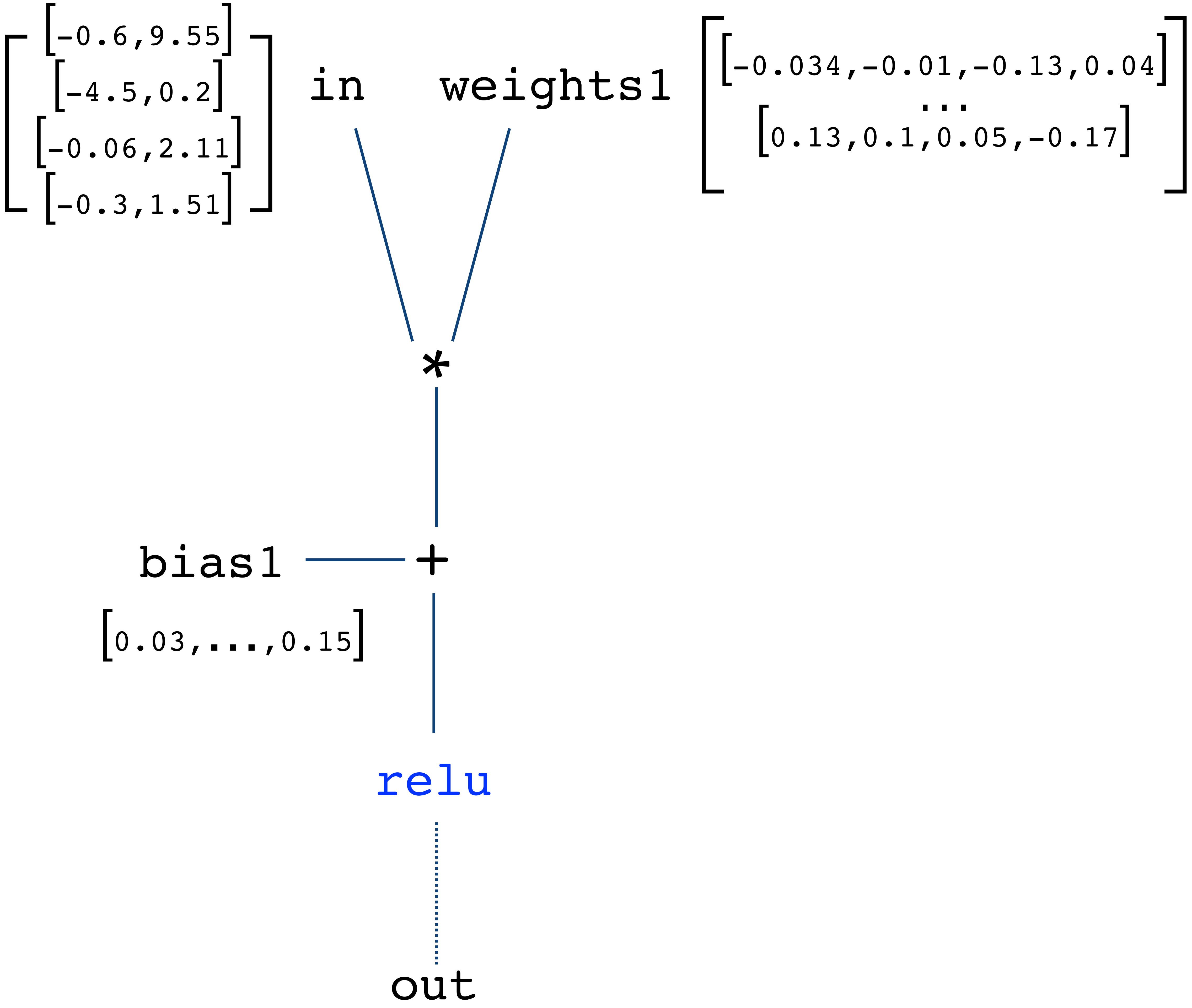


Linearization Step 1: Computing Integer Bits

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



```
res +/- 1e-3
```



Computed integer bits for all variables and constants without overflow

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 2: Exact Linearization of Cost

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

subject to:

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

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

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

non-linear function

Linearization Step 2: Exact Linearization of Cost

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

subject to:

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

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

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

c1: $\gamma_i^{bias} \geq \pi_i^{dot}$

c2: $\gamma_i^{bias} \geq \pi_i^{bias}$

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

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

Optimizing Fractional Bits for Dot and Bias Products

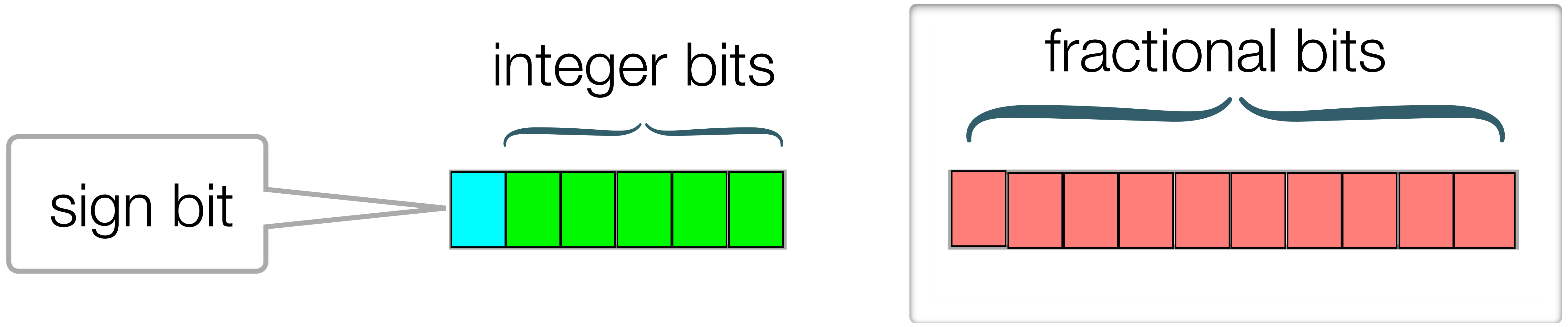
Linearized Problem

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

subject to:

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

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



Optimizing Fractional Bits for Dot and Bias Products

Linearized Problem

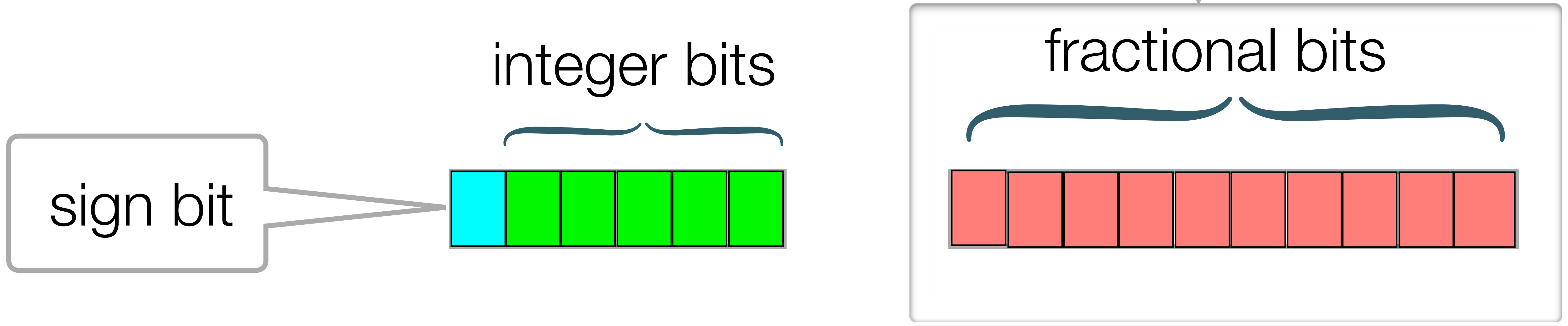
minimize: $\gamma = \sum_{i=1}^{n} \gamma_i^{dot} + \gamma_i^{bias} + \gamma_i^{\alpha}$

subject to:

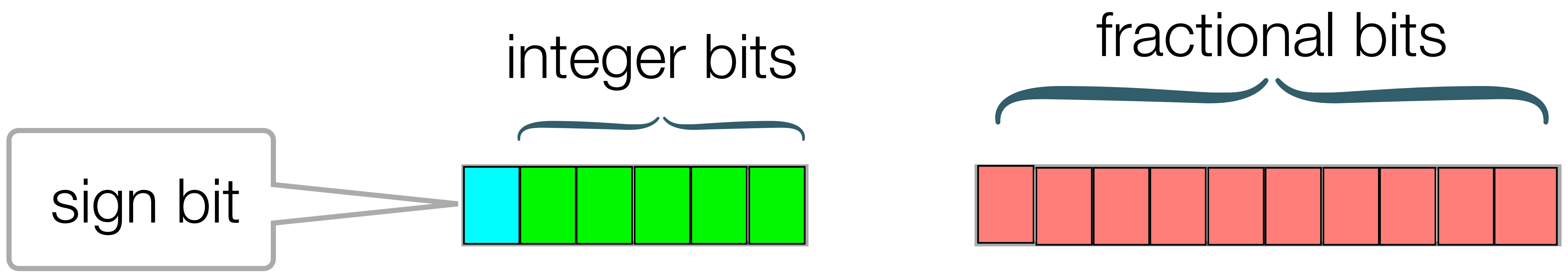
$$\epsilon_n \leq \epsilon_{target}$$

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

MILP solver



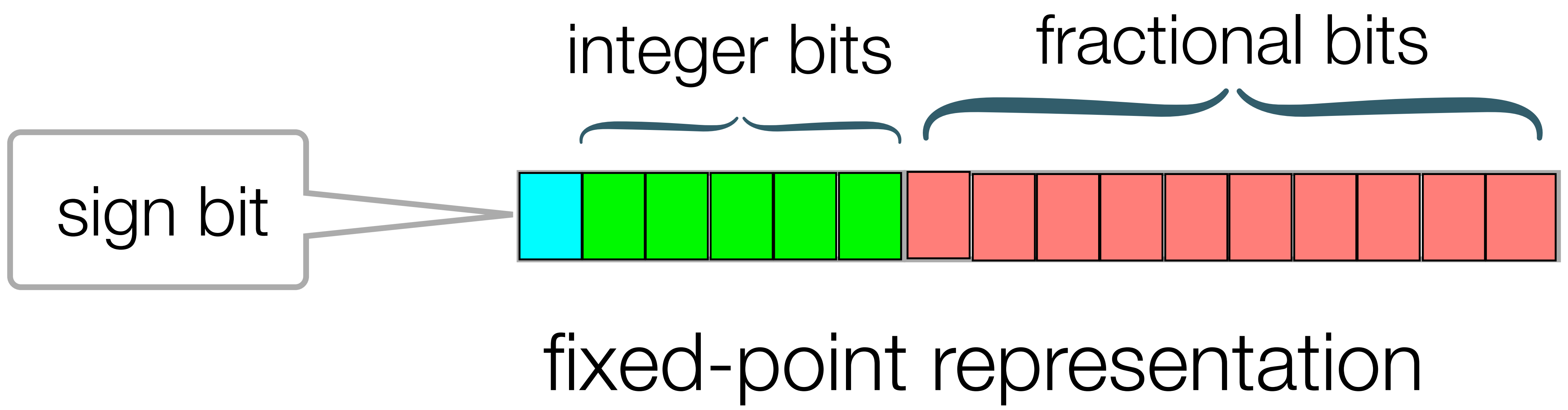
Assign Correctly Rounded Precision to Weights



fixed-point representation

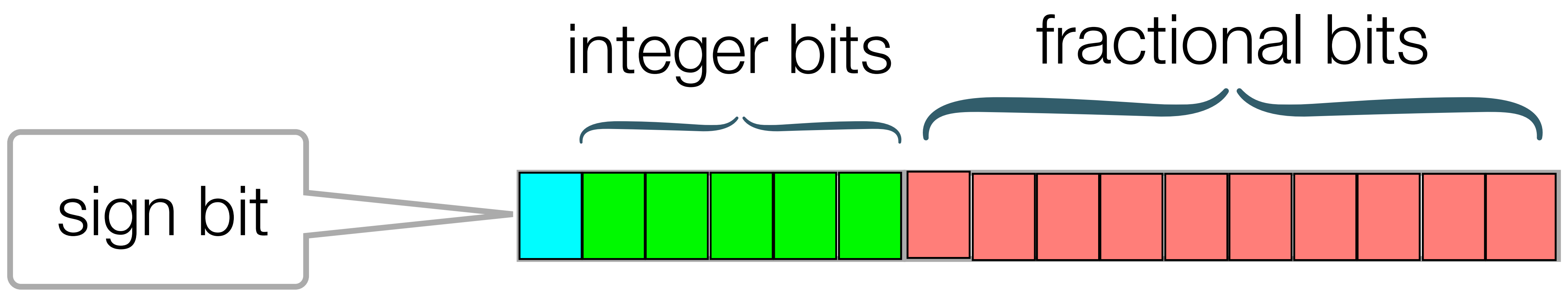
- computed **integer bits** for all variables and constants
- optimized **fractional bits** for dot and bias results assuming precision of weights

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



fixed-point representation

- 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

  using fixed-point sum of products by 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

```
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)
```

high-level model

Aster

Aster: Sound Quantizer for NN Controllers

```
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)
```

high-level model

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<27,1>) (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;
}
```

Aster: Sound Quantizer for NN Controllers

```
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)
```

high-level model

Aster

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<27,1>) (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;
}
```

directly compiled

XILINX

Aster vs State-of-the-Art in terms of Latency

benchmarks	#params	Daisy	Aster
InvPendulum	60		
MountainCar	336		
MPC	720		
DoublePendulum	825		
ACC3	980		
Unicycle	3,500		
Airplane	13,540		
ControllerTora	20,800		
AC8	44,545		

Latencies of implementations considering target error 1e-3

Aster vs State-of-the-Art in terms of Latency

benchmarks	#params	Daisy	Aster
InvPendulum	60	12	
MountainCar	336	27	
MPC	720	inf	
DoublePendulum	825	36	
ACC3	980	49	
Unicycle	3,500	178	
Airplane	13,540	TO	
ControllerTora	20,800	TO	
AC8	44,545	TO	

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

benchmarks	#params	Daisy	Aster
InvPendulum	60	**12**	**12**
MountainCar	336	27	**25**
MPC	720	inf	**35**
DoublePendulum	825	36	**27**
ACC3	980	49	**44**
Unicycle	3,500	178	**27**
Airplane	13,540	TO	
ControllerTora	20,800	TO	
AC8	44,545	TO	

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

benchmarks	#params	Daisy	Aster	
InvPendulum	60	**12**	**12**	
MountainCar	336	27	**25**	
MPC	720	inf	**35**	
DoublePendulum	825	36	**27**	
ACC3	980	49	**44**	
Unicycle	3,500	178	**27**	
Airplane	13,540	TO	unrolled programs are too large!	40 KLOC
ControllerTora	20,800	TO		62 KLOC
AC8	44,545	TO		134 KLOC

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

benchmarks	#params	Daisy	Aster
InvPendulum	60	**12**	**12**
MountainCar	336	27	**25**
MPC	720	inf	**35**
DoublePendulum	825	36	**27**
ACC3	980	49	**44**
Unicycle	3,500	178	**27**
Airplane	13,540	TO	**9,001***
ControllerTora	20,800	TO	**13,158***
AC8	44,545	TO	⊗*

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

benchmarks	Daisy	Aster
InvPendulum	4.19s	
MountainCar	43.68s	
MPC	inf	
DoublePendulum	4m 6.64s	
ACC3	4m 52.05s	
Unicycle	2h 46m 20.65s	
Airplane	TO	
ControllerTora	TO	
AC8	TO	

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

benchmarks	Daisy	Aster
InvPendulum	4.19s	**1.66s**
MountainCar	43.68s	**2.22s**
MPC	inf	**3.50s**
DoublePendulum	4m 6.64s	**3.80s**
ACC3	4m 52.05s	**7.28s**
Unicycle	2h 46m 20.65s	**49.92s**
Airplane	TO	**17m 40.92s**
ControllerTora	TO	**47m 55.95s**
AC8	TO	**3h 49m 31.43s**

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

SCAN ME