Expanding the Horizons of Finite-Precision Analysis

Debasmita Lohar

PhD Defense Talk

27th March, 2024

MAX PLANCK INSTITUTE
FOR SOFTWARE SYSTEMS

UNIVERSITÄT
DES
SAARLANDES

Programming with Finite-Precision

```
def controller(x:Real, y:Real, z:Real): Real = {
  val res = -x*y - 2*y*z - x - z
  return res
}
```

Programming with Finite-Precision

```
          (x:Float32, y:Float32, z:Float32): Float32
def controller(x:Real, y:Real, z:Real): Real = {
  val res = -x*y - 2*y*z - x - z
  return res
}
```

- Reals are implemented in Floating-point / Fixed-point data type

Errors in Finite-Precision

```
def controller(x:Float32, y:Float32, z:Float32): Float32 = {
  val res = -x*y - 2*y*z - x - z
  return res
}      +/- error
```

- Reals are implemented in Floating-point / Fixed-point data type
- Introduces roundoff errors at potentially every operation

Errors in Finite-Precision

```
def controller(x:Float32, y:Float32, z:Float32): Float32 = {
  val res = -x*y - 2*y*z - x - z
  return res
}      +/- error
```

```
0.1 + 0.2 = 0.3
```

real arithmetic

```
>>> 0.1 + 0.2
0.30000000000000004
```

32-bit floating-point arithmetic

Errors in Finite-Precision

```
def controller(x:Float32, y:Float32, z:Float32): Float32 = {
  val res = -x*y - 2*y*z - x - z
  return res
}       +/- error
```

```
0.1 + 0.2 = 0.3
```

real arithmetic

```
>>> 0.1 + 0.2
0.30000000000000004
```

32-bit floating-point arithmetic

Does it even affect real-world systems?

Finite-Precision Errors in Real World

February 1991, Dhahran, Saudi Arabia

Gulf War: Loss of accuracy led to failure in US defense system, 28 killed!

April 1992, Schleswig-Holstein, Germany

Rounding error changed Parliament makeup!

June 1996

Overflow led to explosion of Ariane 5, 39s after lift-off, $370 million lost!

...

Finite-Precision Errors in Real World

February 1991, Dhahran, Saudi Arabia

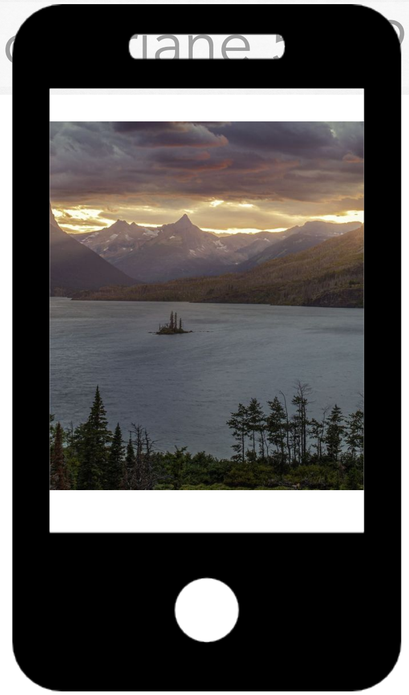
Gulf War: Loss of accuracy led to failure in US defense system, 28 killed!

April 1992, Schleswig-Holstein, Germany

Rounding error changed Parliament makeup!

June 1996

Overflow led to explosion of [illegible] after lift-off, $370 million lost!



May, 2020

Rounding error in luminance computation crashed Andriod phones

Finite-Precision Errors in Real World

February 1991, Dhahran, Saudi Arabia

Gulf War: Loss of accuracy led to failure in US defense system, 28 killed!

April 1992, Schleswig-Holstein, Germany

Rounding error changed Parliament makeup!

June 1996

Overflow led to explosion of Ariane 5, 39s after lift-off, $370 million lost!

...

May 2020

Rounding error in luminance computation crashed Andriod phones

How do we compute the errors?

Finite-Precision Accuracy Analysis

```
(x:Float32, y:Float32, z:Float32): Float32
def controller(x, y, z): = {
  val res = -x*y - 2*y*z - x - z
  return res
} ensuring (res +/- ?)
```

Finite-Precision Accuracy Analysis

```
(x:Float32, y:Float32, z:Float32): Float32
def controller(x, y, z): = {
  val res = -x*y - 2*y*z - x - z
  return res
} ensuring (res +/- ?)
```

compute a bound on the error

Finite-Precision Accuracy Analysis

```
(x:Float32, y:Float32, z:Float32): Float32
def controller(x, y, z): = {
  val res = -x*y - 2*y*z - x - z
  return res
} ensuring (res +/- ?)
```

absolute error:

$$\max_{x,y,z \in I} | f(x,y,z) - \tilde{f}(\tilde{x},\tilde{y},\tilde{z}) |$$

Finite-Precision Accuracy Analysis

$$\max_{x,y,z \in I} | f(x,y,z) - \tilde{f}(\tilde{x},\tilde{y},\tilde{z}) |$$

worst-case error analysis for small programs

Daisy FLUCTUAT Rosa

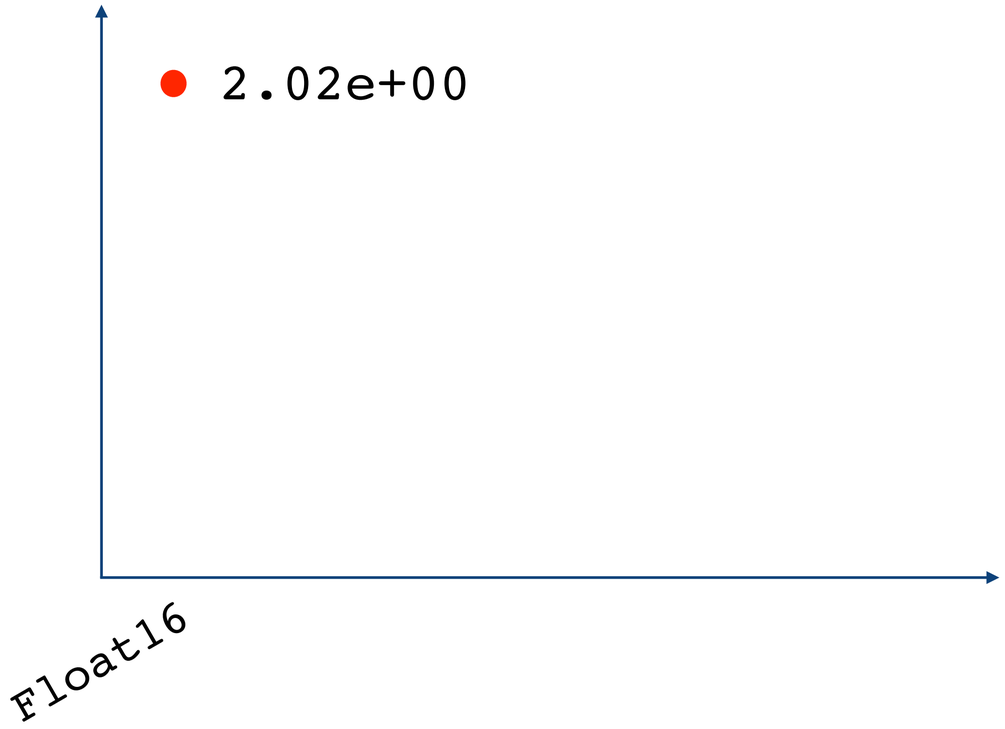
FPTaylor PRECiSA ···

Errors depend on Precision used

```
(x:Float16, y:Float16, z:Float16)
def controller(x, y, z): ___ = {
  val res = -x*y - 2*y*z - x - z
  return res
} ensuring (res +/- ?)
```

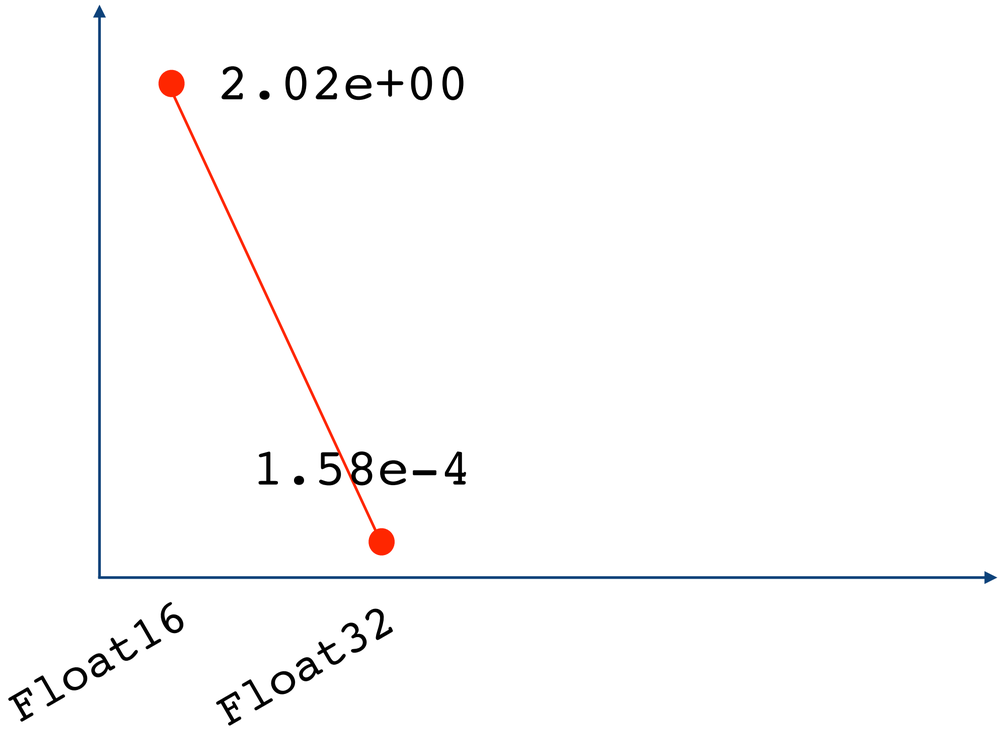
Errors depend on Precision used

```
(x:Float16, y:Float16, z:Float16)
def controller(x, y, z): ___ = {
  val res = -x*y - 2*y*z - x - z
  return res
} ensuring (res +/- ?)
```



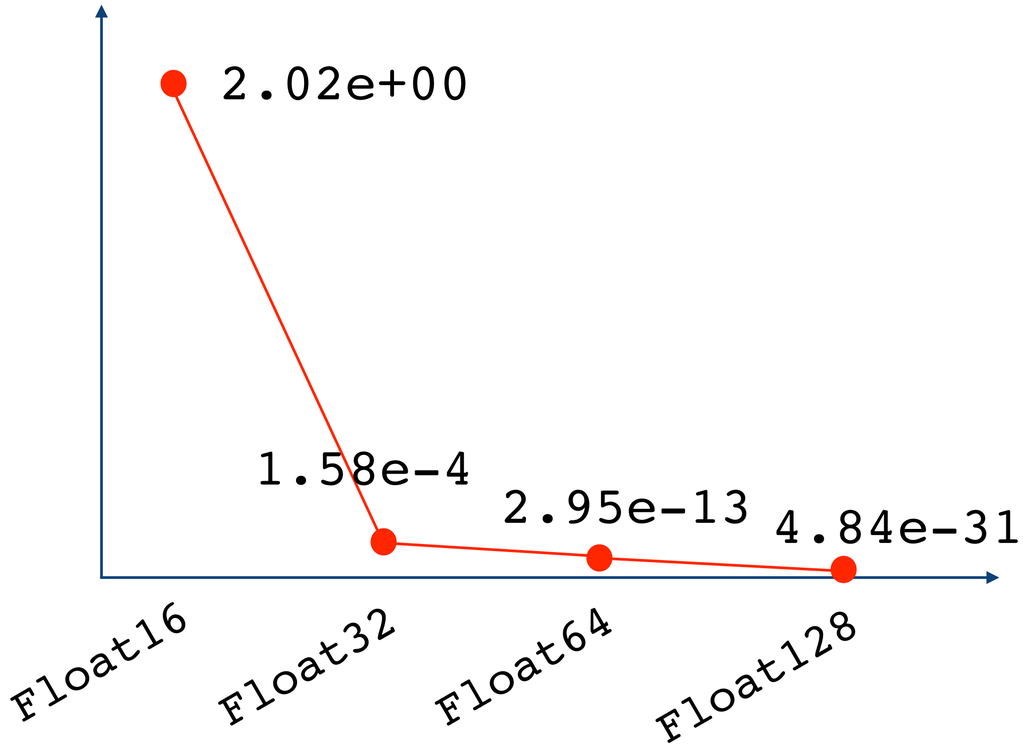
Errors depend on Precision used

```
(x:__, y:__, z:__)
def controller(x, y, z): ___ = {
  val res = -x*y - 2*y*z - x - z
  return res
} ensuring (res +/- ?)
```



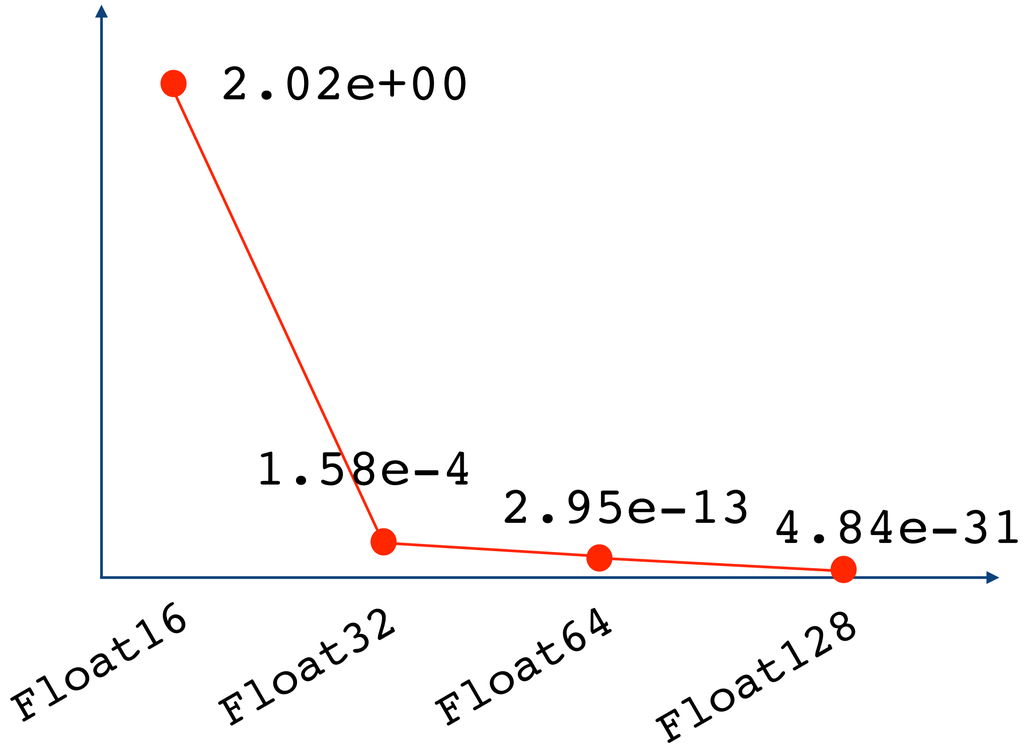
Errors depend on Precision used

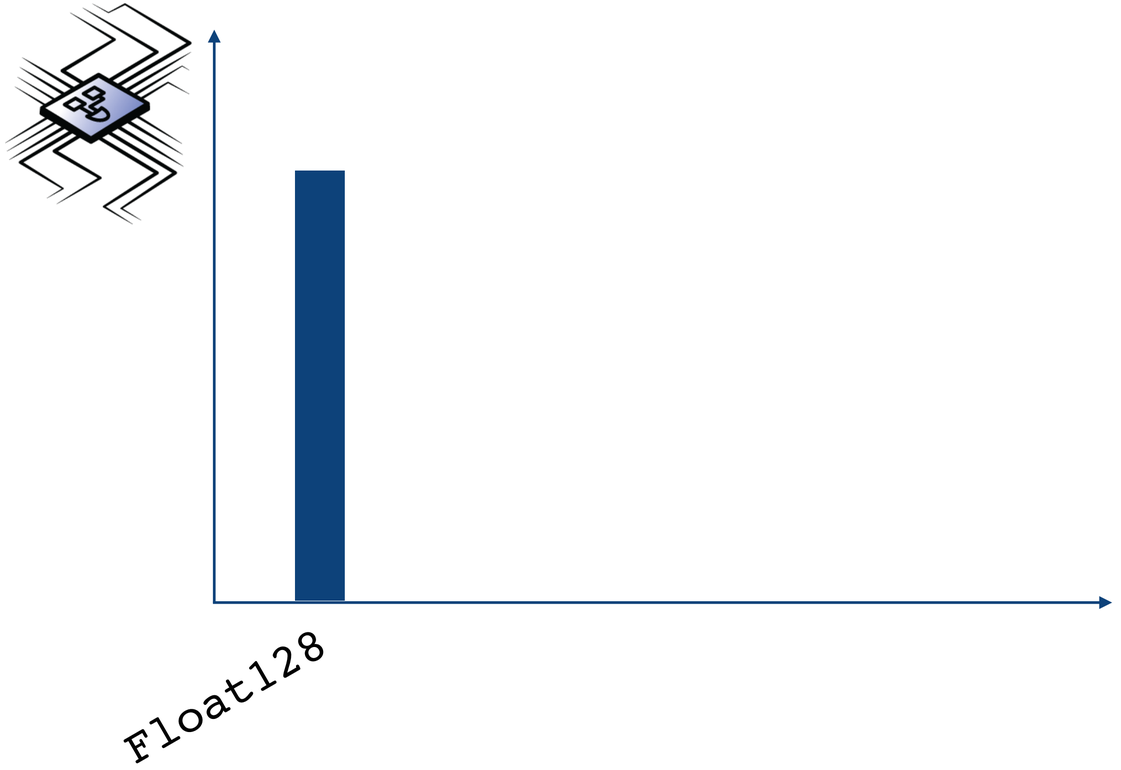
```
(x:↑__, y:↑__, z:↑__)
def controller(x, y, z): ___ = {
  val res = -x*y - 2*y*z - x - z
  return res
} ensuring (res +/- ?)
```



So are the Resource Costs!

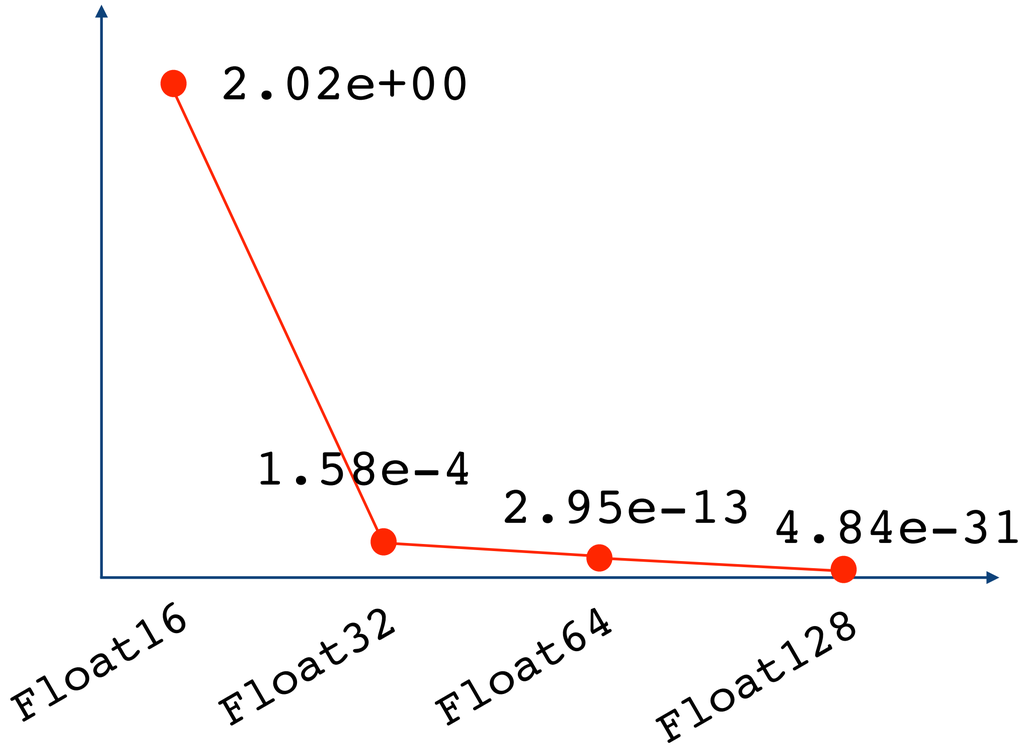
```
(x:↑__, y:↑__, z:↑__)
def controller(x, y, z): ___ = {
  val res = -x*y - 2*y*z - x - z
  return res
} ensuring (res +/- ?)
```

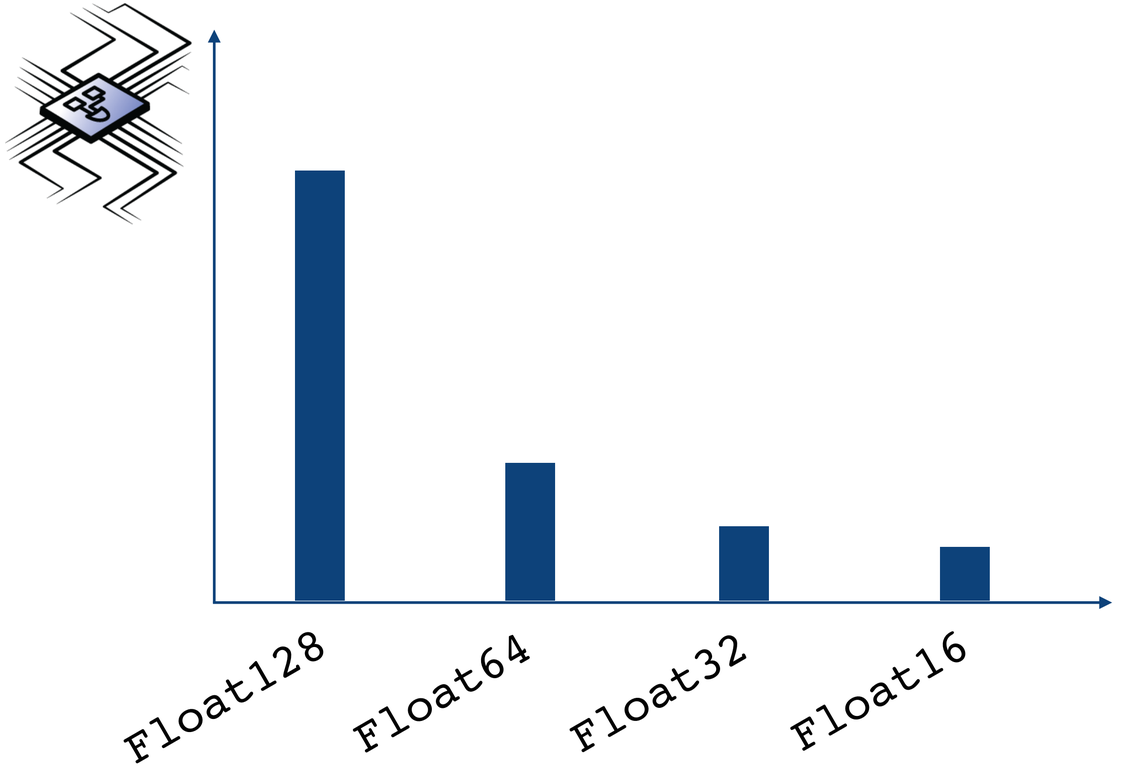




So are the Resource Costs!

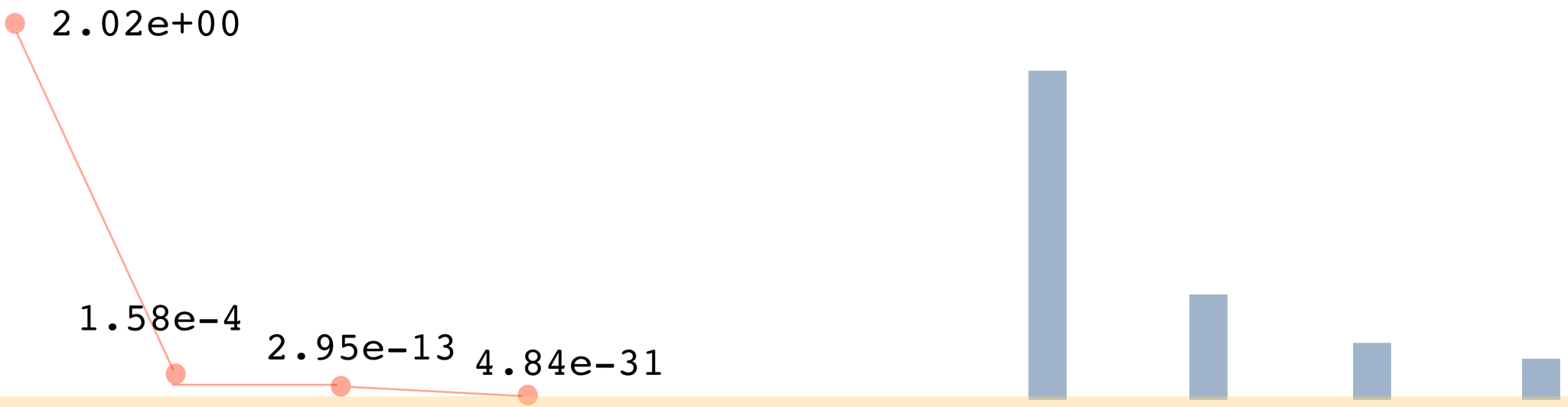
```
(x:↑__, y:↑__, z:↑__)
def controller(x, y, z): ___ = {
  val res = -x*y - 2*y*z - x - z
  return res
} ensuring (res +/- ?)
```





So are the Resource Costs!

```
(x:__, y:__, z:__)
def controller(x, y, z): ___ = {
  val res = -x*y - 2*y*z - x - z
  return res
} ensuring (res +/- ?)
```



We need to find a tradeoff between accuracy and resources!

Finite-Precision Optimization

```
(x:Float32, y:Float32, z:Float32)
 def controller(x, y, z): ___ = {
   val res = -x*y - 2*y*z - x - z
   return res
 } ensuring (res +/- ?)
```

```
def controller(x: ?, y: ?, z: ?): ? = {
  val res = -x*y - 2*y*z - x - z
  return res
} ensuring res +/- 0.00197
```

find the lowest precision satisfying error bound

Finite-Precision Optimization

```
(x:Float32, y:Float32, z:Float32)
 def controller(x, y, z): ___ = {
   val res = -x*y - 2*y*z - x - z
   return res
 } ensuring (res +/- ?)
```

```
def controller(x: ?, y: ?, z: ?): ? = {
  val res = -x*y - 2*y*z - x - z
  return res
} ensuring res +/- 0.00197
```

mixed-precision optimization

- minimize resource cost still satisfying the error
- assign different precisions to different variables

Finite-Precision Optimization

```
(x:Float32, y:Float32, z:Float32)
 def controller(x, y, z): ___ = {
   val res = -x*y - 2*y*z - x - z
   return res
 } ensuring (res +/- ?)
```

```
def controller(x: ?, y: ?, z: ?): ? = {
  val res = -x*y - 2*y*z - x - z
  return res
} ensuring res +/- 0.00197
```

- minimize resource cost still satisfying the error
- assign different precisions to different variables

worst-case tuning for small (floating-point) programs

Daisy FPTuner

The Horizons of Finite-Precision Analysis

Accuracy Analysis

Optimization

worst-case error analysis for small programs

Daisy FLUCTUAT Rosa

FPTaylor PRECiSA ...

worst-case tuning for small (floating-point) programs

Daisy FPTuner

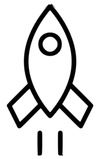
Our Work: Extending the Horizon of Finite-Precision Analysis

Accuracy Analysis

considering probability distribution of inputs

iFM '19 EMSOFT '18

Probabilistic Analysis



~~worst-case error~~ analysis for small programs

Daisy FLUCTUAT Rosa

FPTaylor PRECiSA ...

Optimization

worst-case tuning for small (floating-point) programs

Daisy FPTuner

Our Work: Extending the Horizon of Finite-Precision Analysis

Accuracy Analysis

Optimization

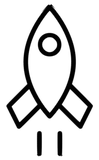
handling larger programs

iFM '19 EMSOFT '18

Probabilistic Analysis

TACAS '21

Static + Dynamic Analysis



worst-case error analysis for ~~small programs~~

Daisy FLUCTUAT Rosa

FPTaylor PRECiSA ...

worst-case tuning for small (floating-point) programs

Daisy FPTuner

Our Work: Extending the Horizon of Finite-Precision Analysis

Accuracy Analysis

Optimization

iFM '19 EMSOFT '18

Probabilistic Analysis

TACAS '21

Static + Dynamic Analysis

specializing mixed fixed tuning for NNs

EMSOFT '23

NN Quantization

worst-case error analysis for small programs

Daisy FLUCTUAT Rosa

FPTaylor PRECiSA ...

worst-case tuning for ~~small (floating-point) programs~~

Daisy FPTuner

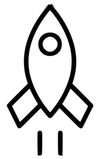
Today's Talk: Probabilistic Error Analysis and NN Quantization

Accuracy Analysis

Optimization

iFM '19 EMSOFT '18

Probabilistic Analysis



TACAS '21

Static + Dynamic Analysis

EMSOFT '23

NN Quantization

~~worst-case error~~ analysis for small programs

Daisy FLUCTUAT Rosa

FPTaylor PRECiSA ...

worst-case tuning for ~~small (floating-point) programs~~

Daisy FPTuner

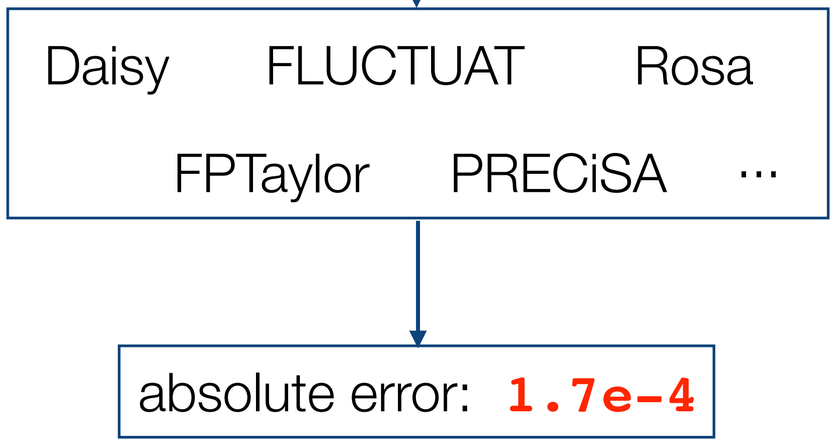
Probabilistic Roundoff Error Analysis

iFM'19

How do we take into account uncertainties in the inputs and compute the distribution of errors at the output?

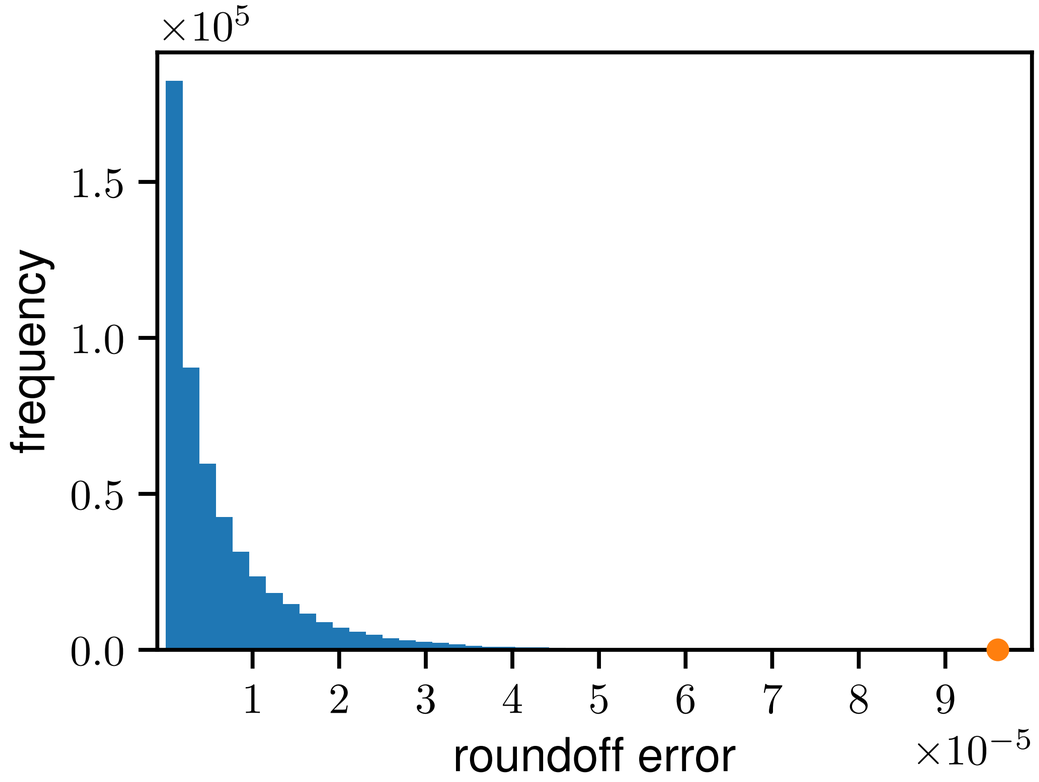
State-of-the-Art: Worst-Case Error Analysis

```
def controller(x:Float32, y:Float32, z:Float32): Float32 = {
  require (-15.0 <= x, y, z <= 15.0)
  val res = -x*y - 2*y*z - x - z
    return res
} ensuring (res +/- ?)
```



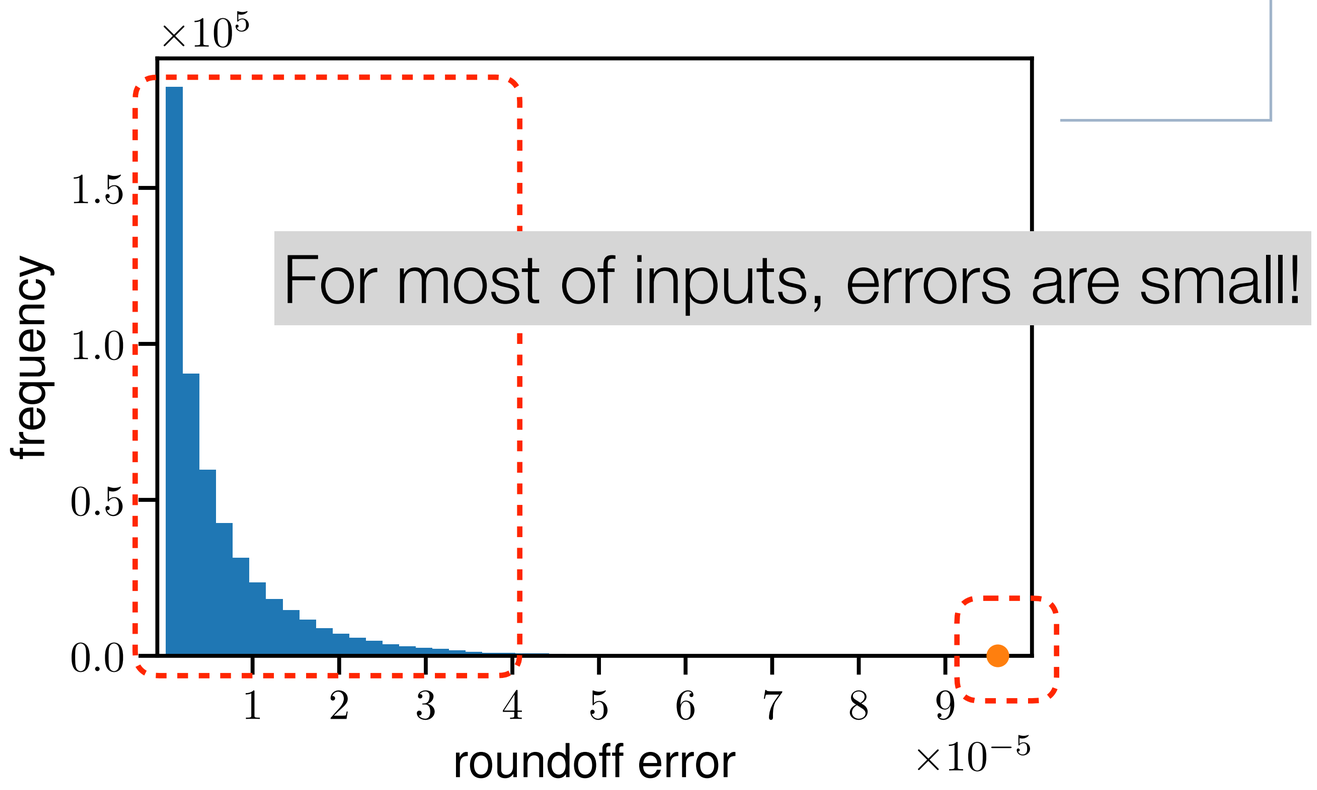
Worst-case can be pessimistic!

```
def controller(x:Float32, y:Float32, z:Float32): Float32 = {
  require (-15.0 <= x, y, z <= 15.0)
  val res = -x*y - 2*y*z - x - z
    return res
} ensuring (res +/- ?)
```



Worst-case can be pessimistic!

```
def controller(x:Float32, y:Float32, z:Float32): Float32 = {
  require (-15.0 <= x, y, z <= 15.0)
  val res = -x*y - 2*y*z - x - z
    return res
} ensuring (res +/- ?)
```



Scenario 1: Applications may tolerate large infrequent errors

```
def controller(x:Float32, y:Float32, z:Float32): Float32 = {
 require (-15.0 <= x, y, z <= 15.0)
 val res = -x*y - 2*y*z - x - z
   return res
}ensuring (error <= 1.5e-4, 0.85)
```

tolerates big errors occurring with <= **0.15** probability

Scenario 1: Applications may tolerate large infrequent errors

```
(x:Float64, y:Float64, z:Float64): Float64
```

```
def controller(x:Float32, y:Float32, z:Float32): Float32 = {
 require (-15.0 <= x, y, z <= 15.0)
 val res = -x*y - 2*y*z - x - z
   return res
}ensuring (error <= 1.5e-4, 0.85)
```

worst-case error: **1.7e-4**

tolerates big errors occurring with <= **0.15** probability

Scenario 1: Applications may tolerate large infrequent errors

```
(x:Float64, y:Float64, z:Float64): Float64
```

```
def controller(x:Float32, y:Float32, z:Float32): Float32 = {
 require (-15.0 <= x, y, z <= 15.0)
 val res = -x*y - 2*y*z - x - z
   return res
}ensuring (error <= 1.5e-4, 0.85)
```

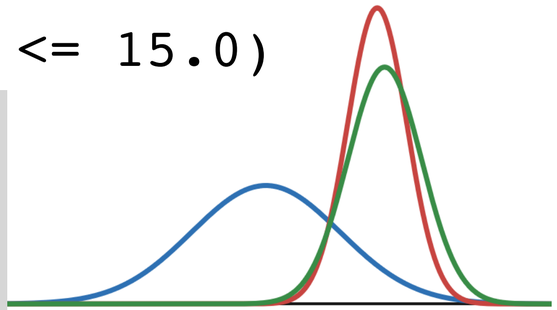
worst-case error: **1.7e-4**

tolerates big errors occurring with <= **0.15** probability

We need to analyze roundoff errors probabilistically!

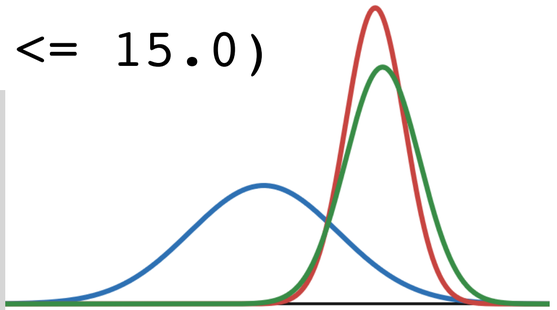
Our Contribution: Probabilistic Analysis for Roundoff Errors

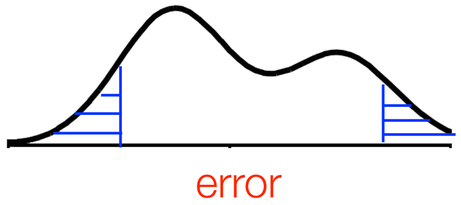
```
def controller(x:Float32, y:Float32, z:Float32): Float32 = {
require (-15.0 <= x, y, z <= 15.0)
  x:= gaussian(4.0, 0.5)
  y:= gaussian(4.75, 0.2)
  z:= gaussian(4.8, 0.25)
   val res = -x*y - 2*y*z - x - z
     return res
} ensuring (error <= 1.5e-4, 0.85)
```



Our Contribution: Probabilistic Analysis for Roundoff Errors

```
def controller(x:Float32, y:Float32, z:Float32): Float32 = {
require (-15.0 <= x, y, z <= 15.0)
 x:= gaussian(4.0, 0.5)
 y:= gaussian(4.75, 0.2)
 z:= gaussian(4.8, 0.25)
  val res = -x*y - 2*y*z - x - z
    return res
} ensuring (error <= 1.5e-4, 0.85)
```

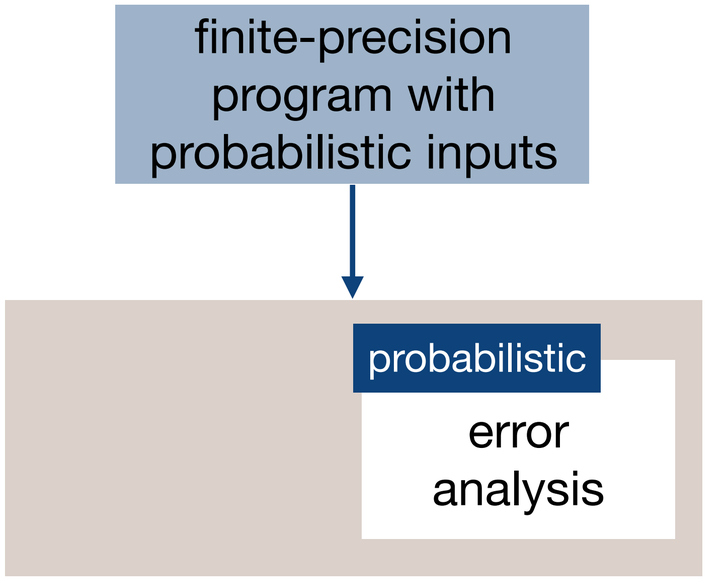




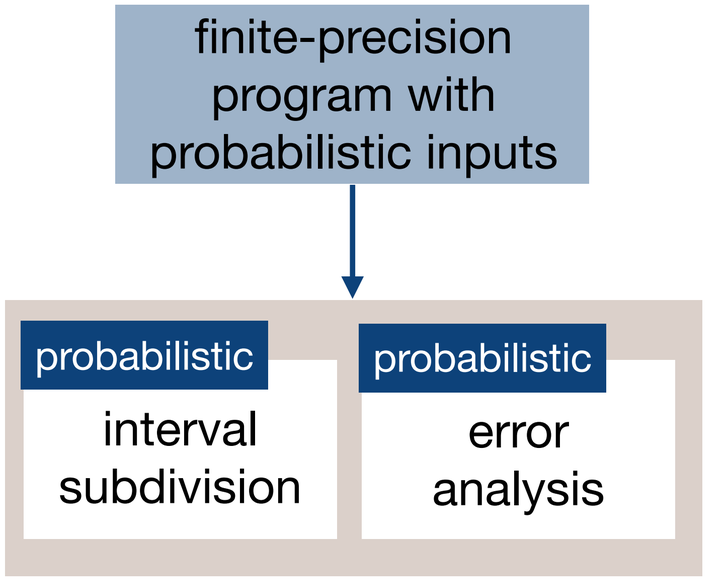
✓ probability distribution of errors

✓ a refined error that occurs with the threshold probability

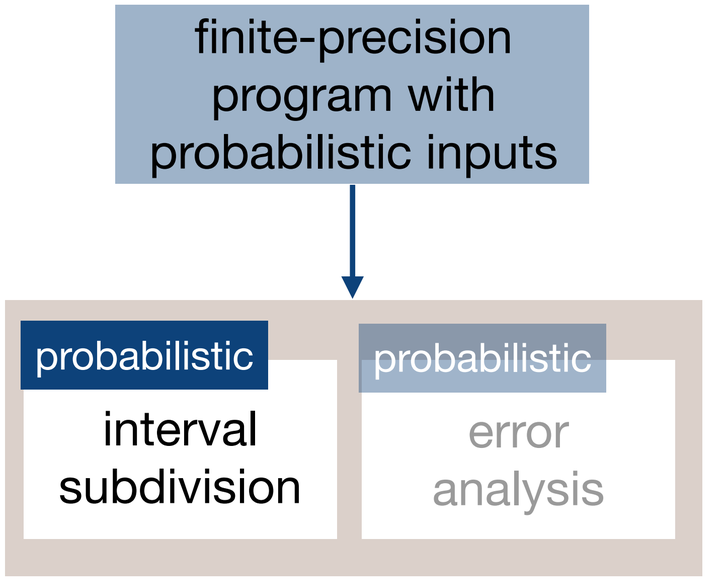
Overview: Sound Probabilistic Roundoff Error Analysis



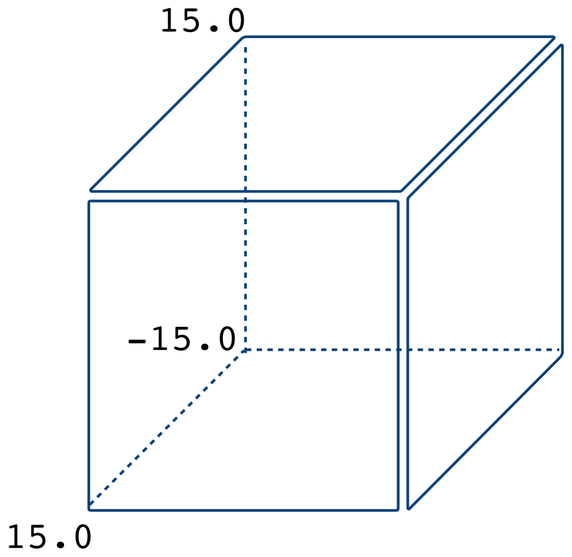
Overview: Sound Probabilistic Roundoff Error Analysis

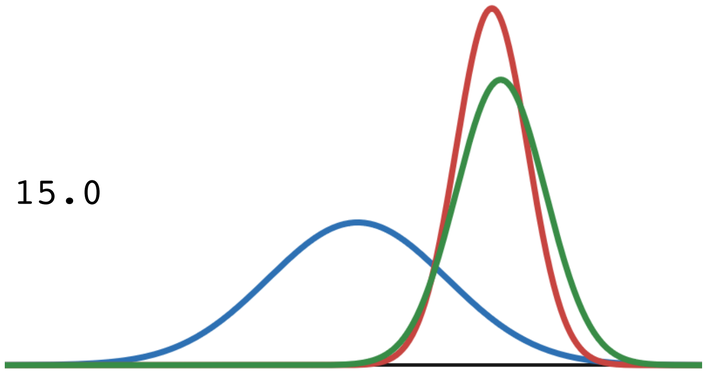


Probabilistic Interval Subdivision

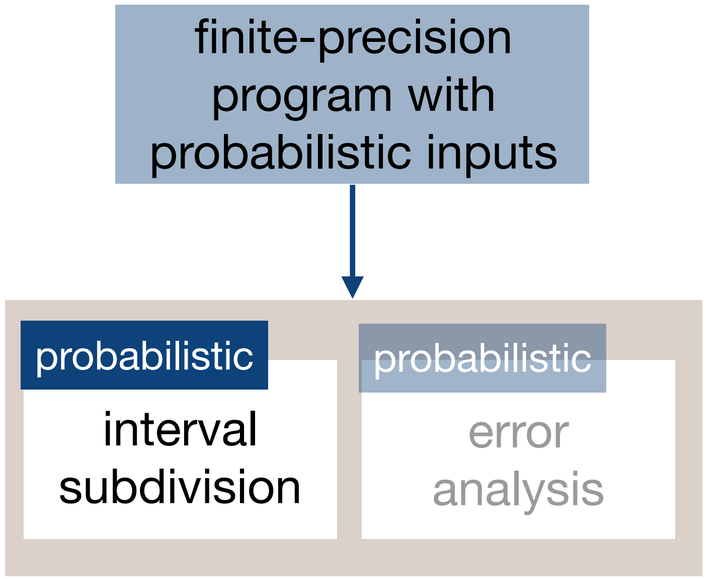



```
require (-15.0 <= x, y, z <= 15.0)
```

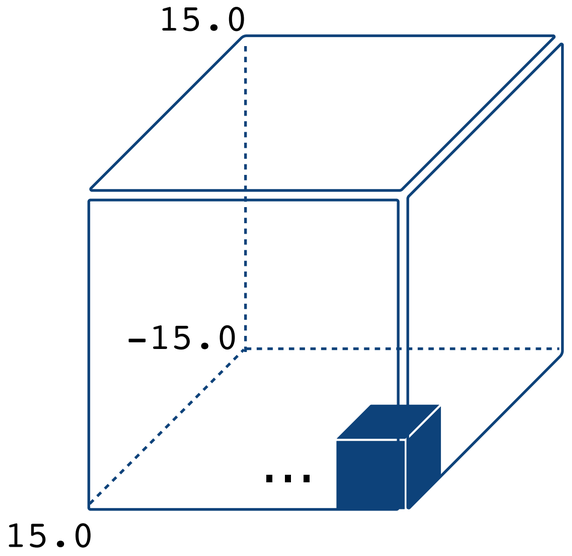


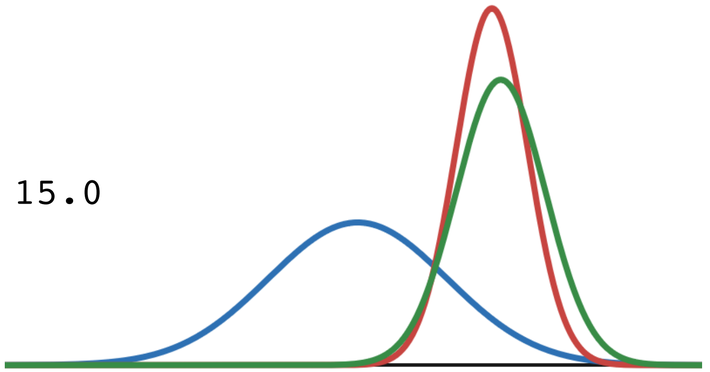


Probabilistic Interval Subdivision

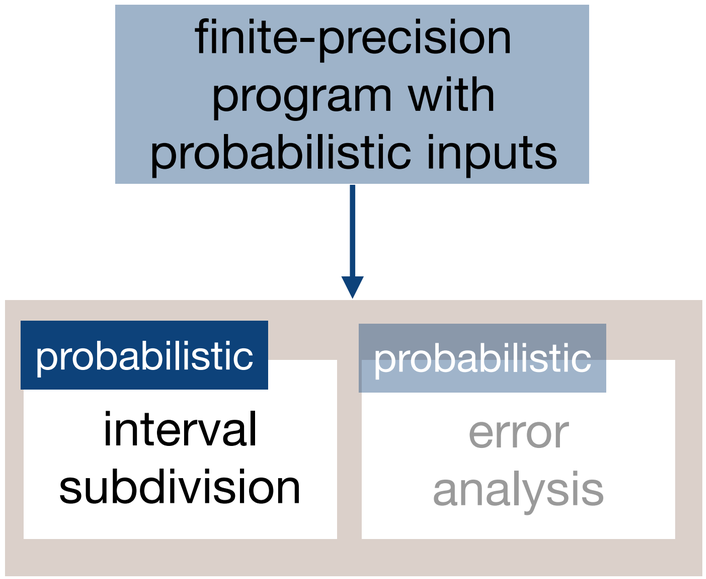



```
require (-15.0 <= x, y, z <= 15.0)
```

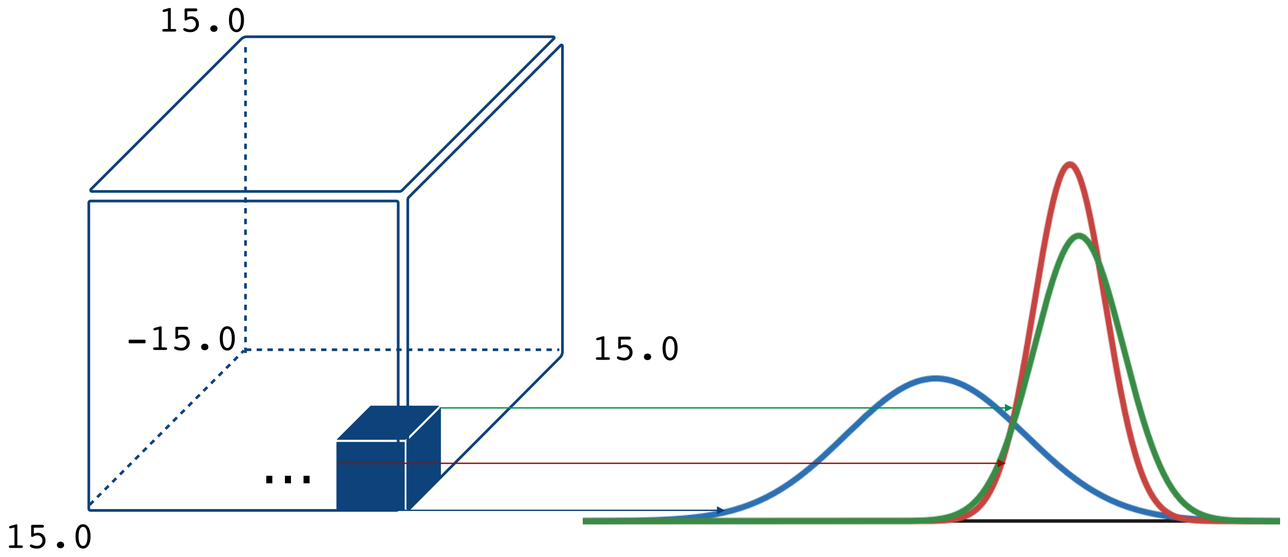




Probabilistic Interval Subdivision



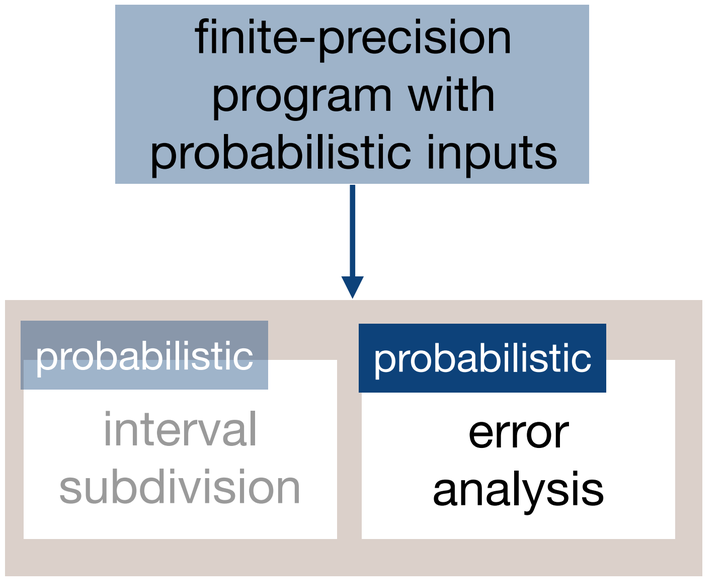

```
require (-15.0 <= x, y, z <= 15.0)
```

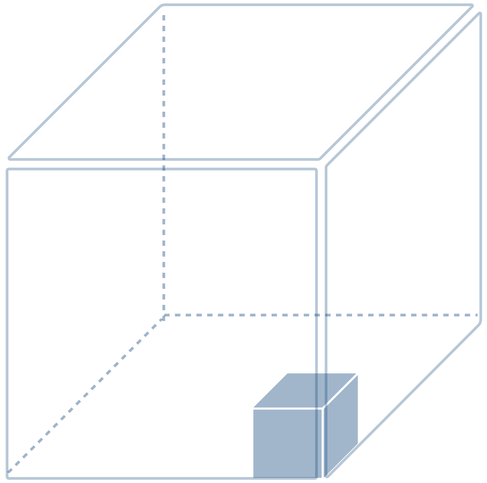


subdomain with a probability taking Cartesian Product:

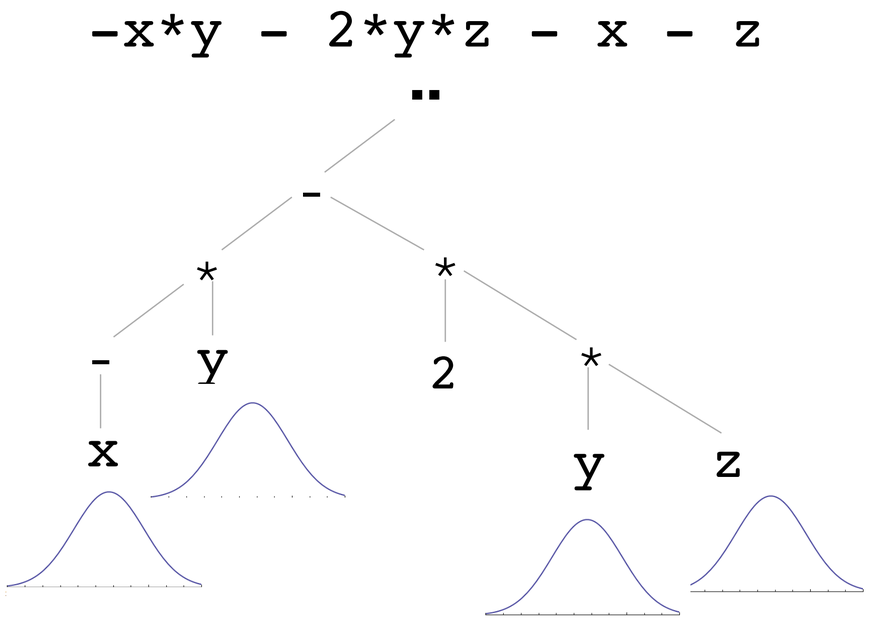
$$\forall i \in x, \forall j \in y, \forall k \in z, p_{ijk} = x_i \times y_j \times z_k$$

Probabilistic Error Analysis





$< s_{ijk}, p_{ijk} >$

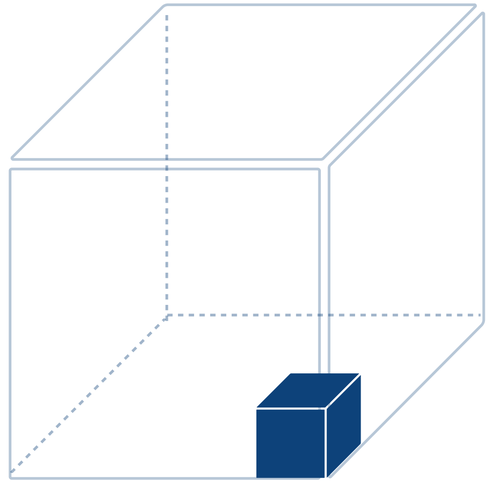


Probabilistic Error Analysis

finite-precision program with probabilistic inputs

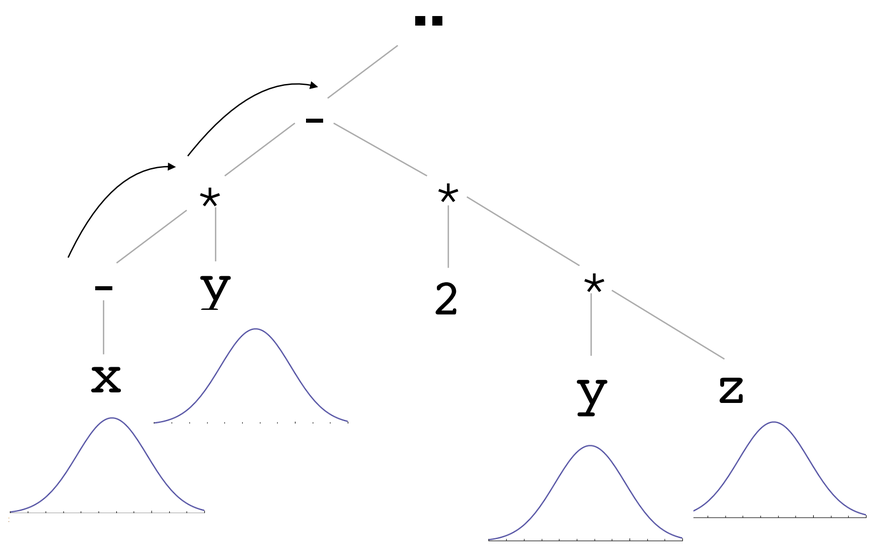
probabilistic interval subdivision

probabilistic error analysis

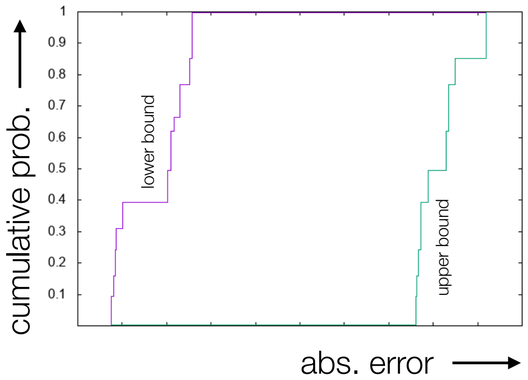


$< s_{ijk}, p_{ijk} >$

```
-x*y - 2*y*z - x - z
```



error distribution:

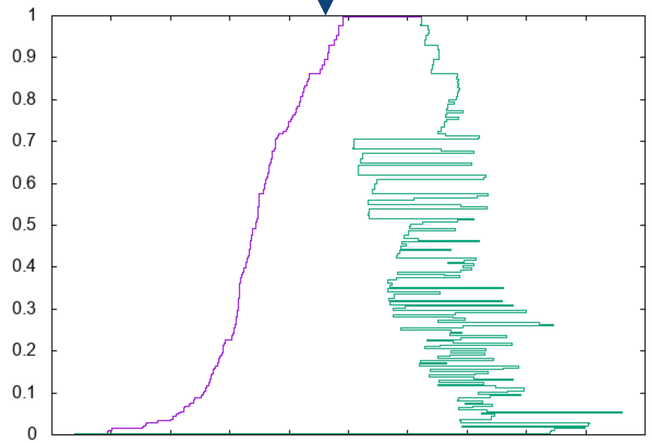


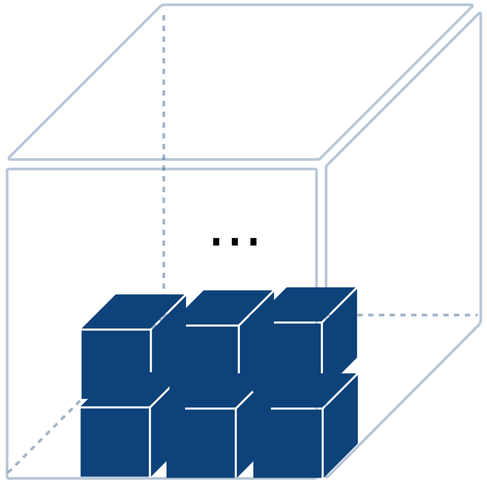
Probability Distribution of Errors

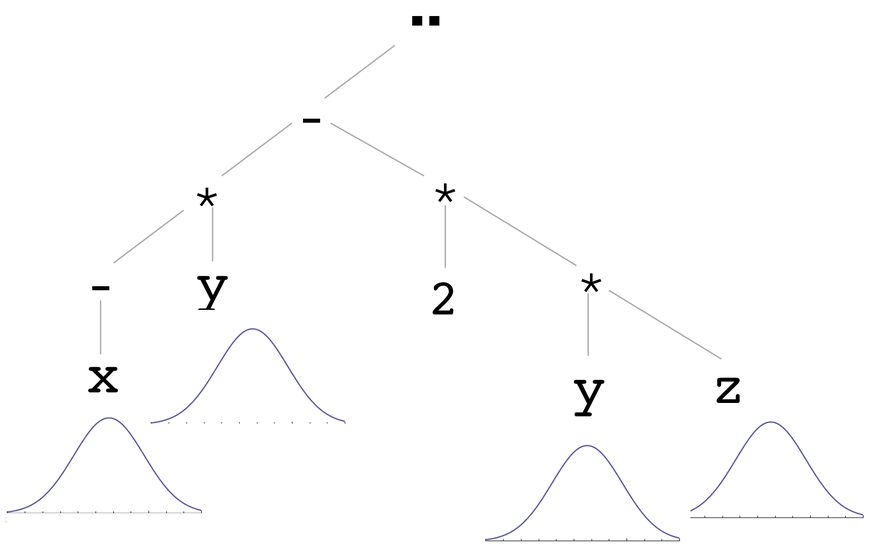
finite-precision program with probabilistic inputs

probabilistic interval subdivision

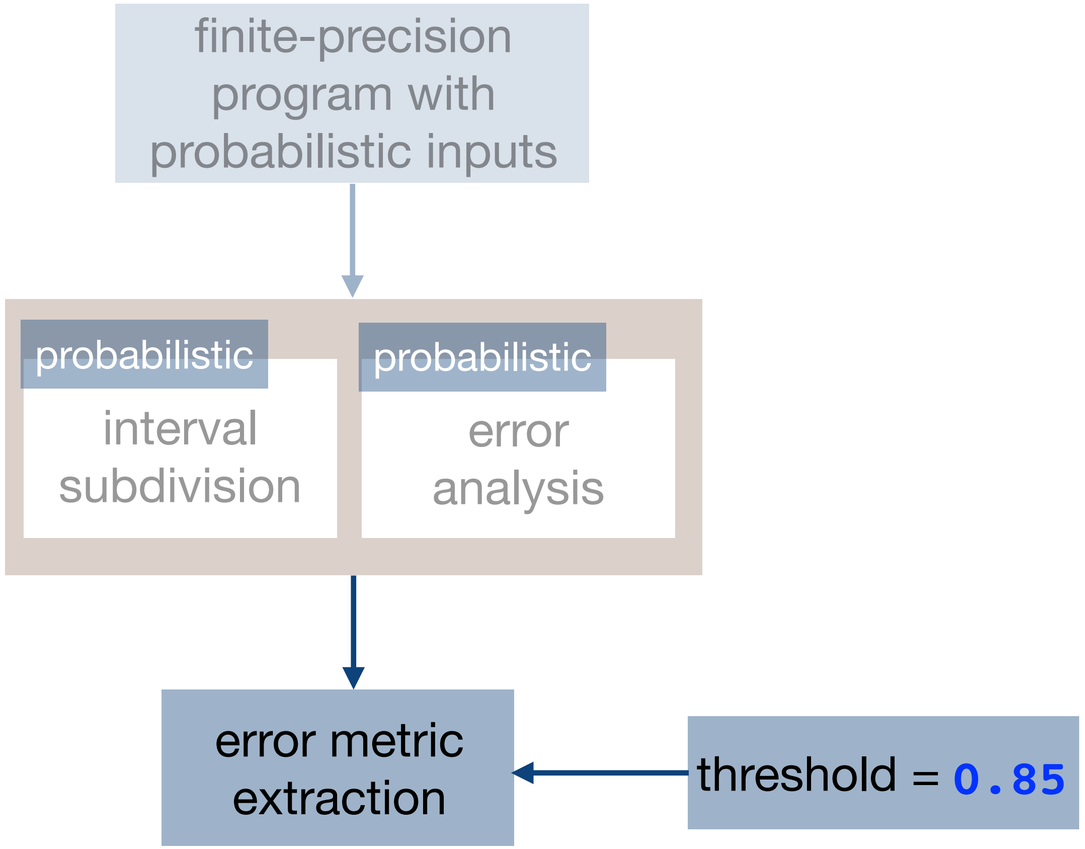
probabilistic error analysis

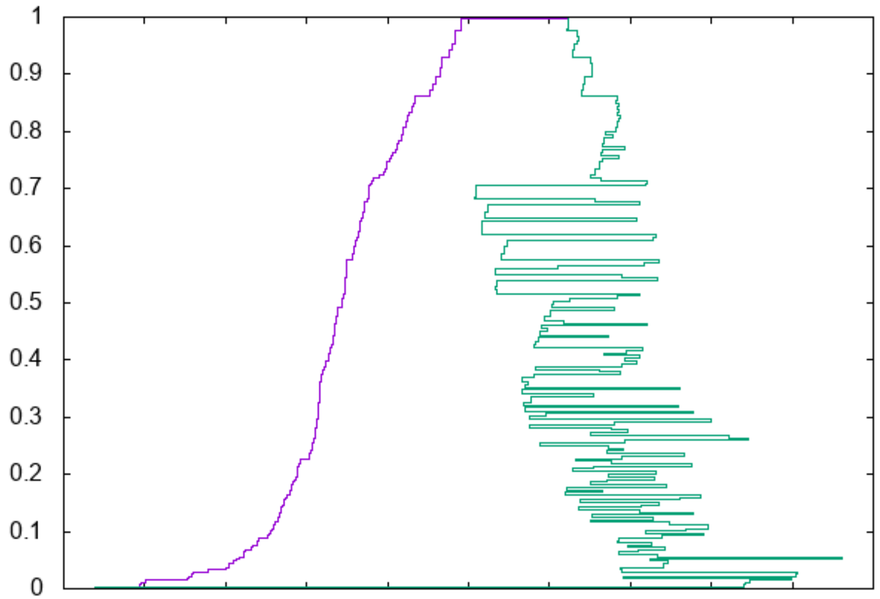




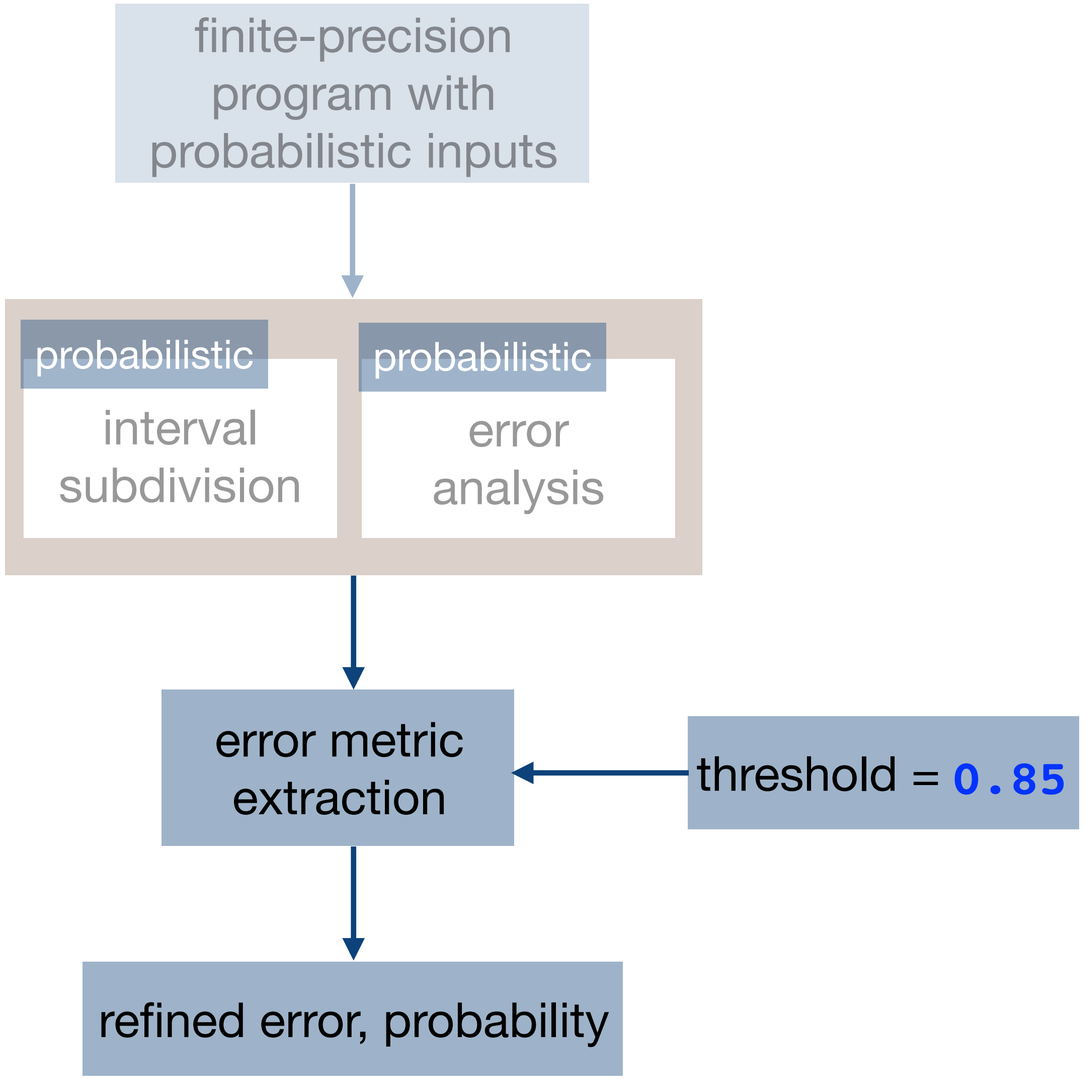


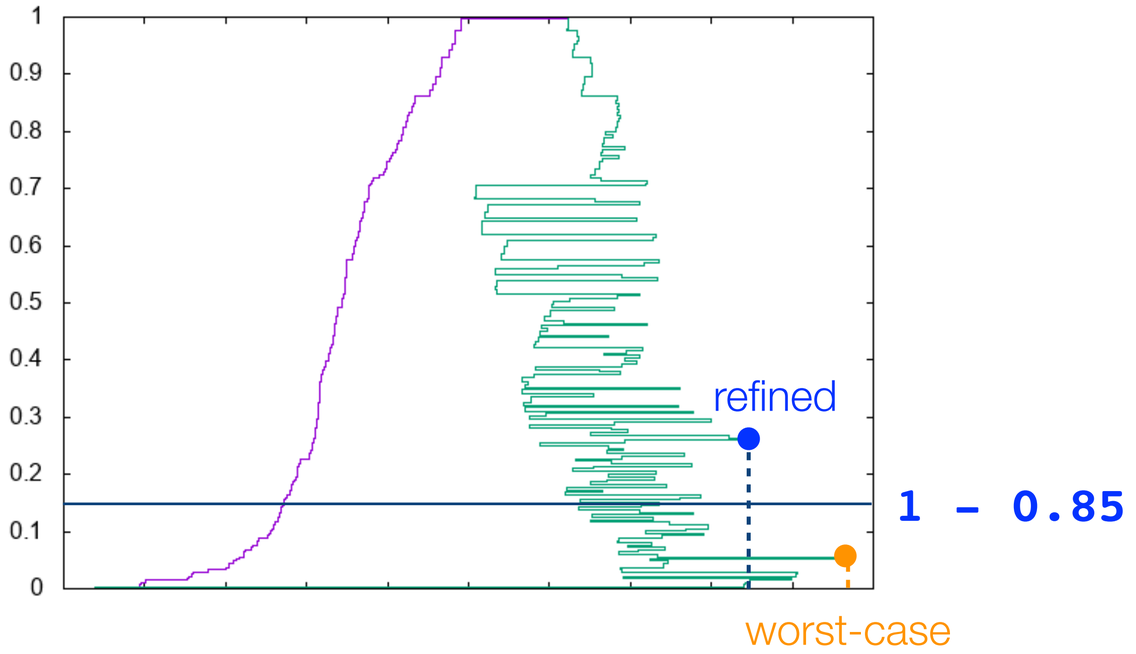
Refined Error Bounds





Refined Error Bounds





Summary of Results: Probabilistic Error Analysis

threshold probability: `0.85`, `32`-bit floating-point error

#benchmarks	#inputs	#arith-ops
25	1 – 9	4 – 25

Summary of Results: Probabilistic Error Analysis

threshold probability: `0.85`, `32`-bit floating-point error

#benchmarks	#inputs	#arith-ops	error reduction (%) from worst-case to the largest frequent			
			average		max	
			gaussian	uniform	gaussian	uniform
25	1 - 9	4 - 25	17.0	16.2	48.9	45.1

Summary of Results: Probabilistic Error Analysis

threshold probability: `0.85`, `32`-bit floating-point error

#benchmarks	#inputs	#arith-ops	error reduction (%) from worst-case to the largest frequent			
			average		max	
			gaussian	uniform	gaussian	uniform
25	1 - 9	4 - 25	17.0	16.2	48.9	45.1

Reductions up to 73.1%
with approximate hardware specifications!

Takeaways

- Not all applications need worst-case guarantees
- Providing bounds on most frequent errors can be resource-efficient
- An automated probabilistic error analyzer: PrAn  
  - strikes a balance between accuracy and complexity
  - handles different distributions, dependencies, and thresholds

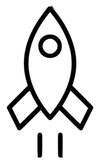
Today's Talk: Probabilistic Error Analysis and NN Quantization

Accuracy Analysis

Optimization

iFM '19 EMSOFT '18

Probabilistic Analysis



TACAS '21

Static + Dynamic Analysis

EMSOFT '23

NN Quantization

~~worst-case error~~ analysis for small programs

Daisy FLUCTUAT Rosa

FPTaylor PRECiSA ...

worst-case tuning for ~~small (floating-point) programs~~

Daisy FPTuner

Sound Mixed Fixed-Point Quantization of Neural Networks

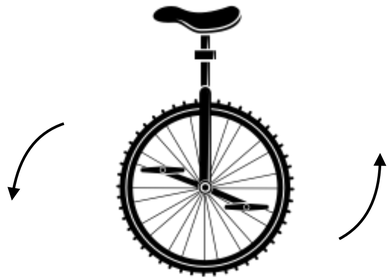
EMSOFT'23

How do we generate quantized implementations for neural networks that meet specified worst-case error bounds?

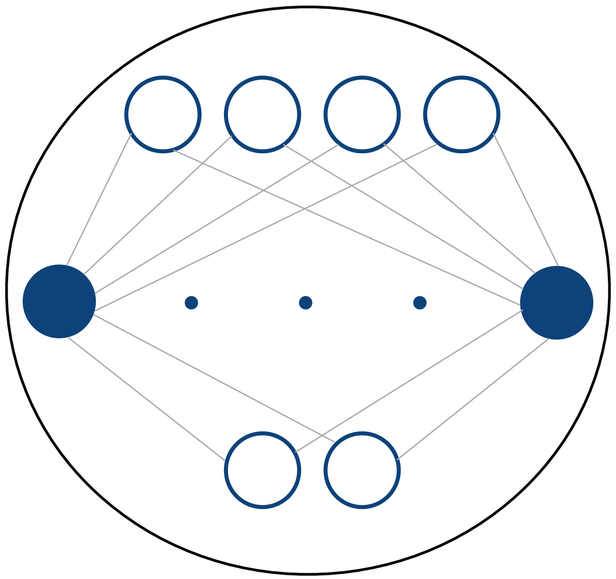
Neural networks are ubiquitous in safety-critical systems!

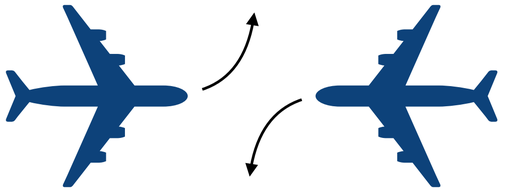
Adaptive Cruise Control

...



Unicycle Controller



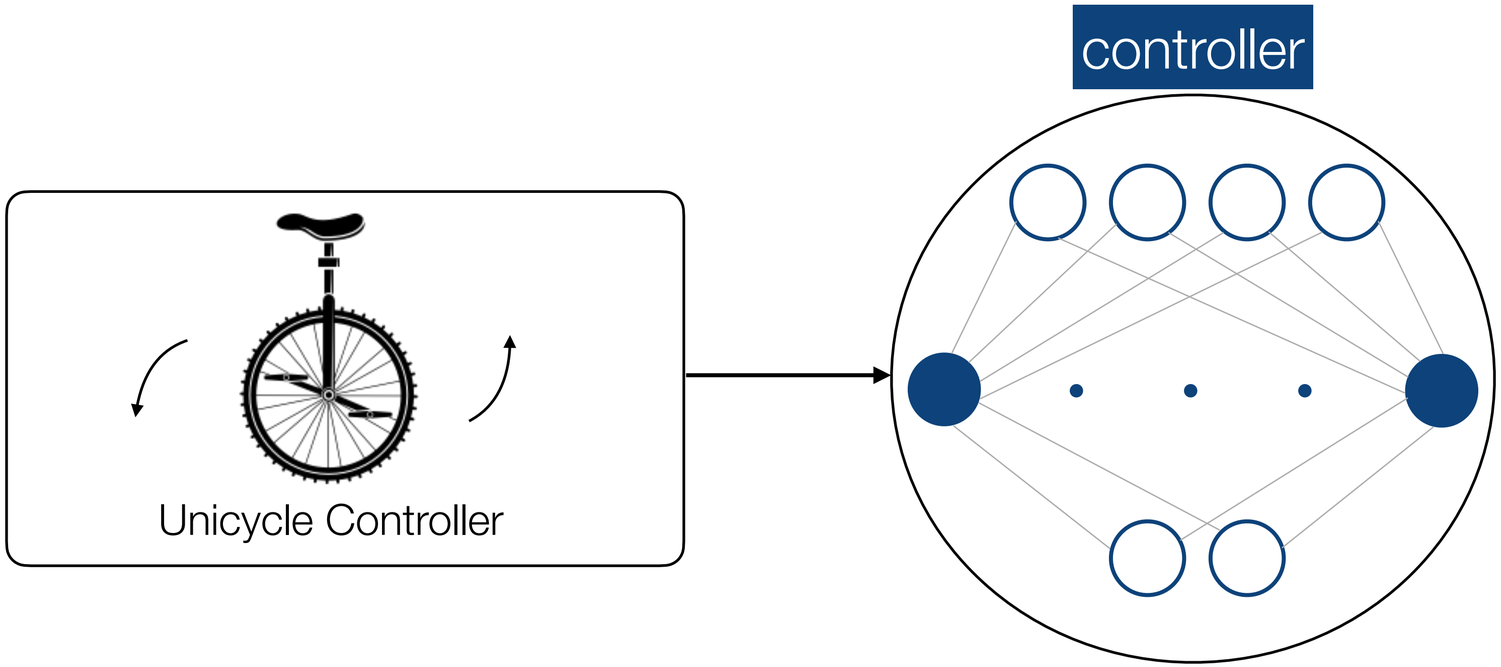


Collision Avoidance System

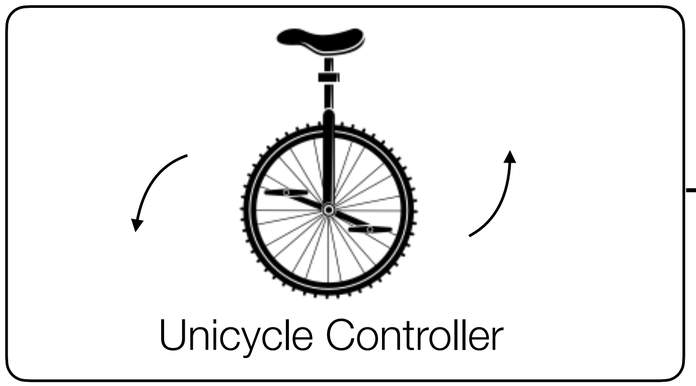
...

Drone Controller

Neural Networks as Controllers



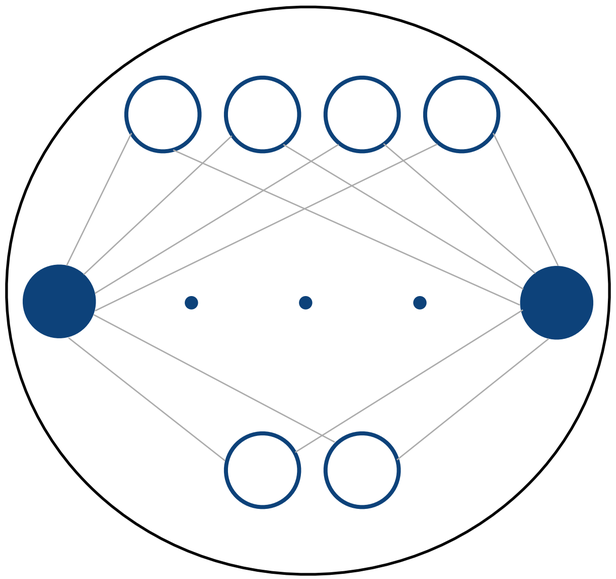
Neural Networks as Controllers




```
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
}
```

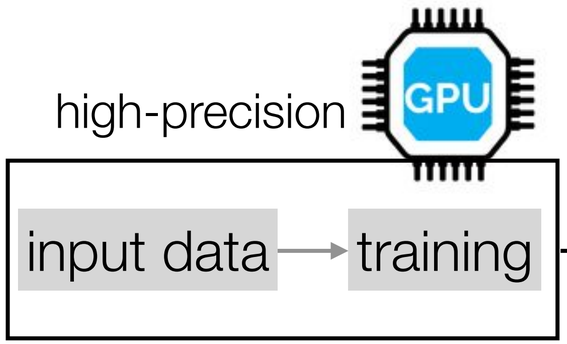
feed-forward regression models

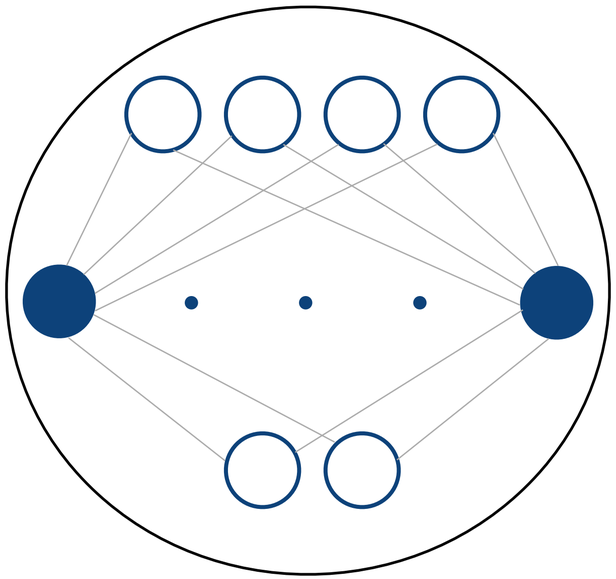
Models are trained in High-Precision



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

Models are trained in High-Precision

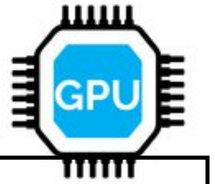




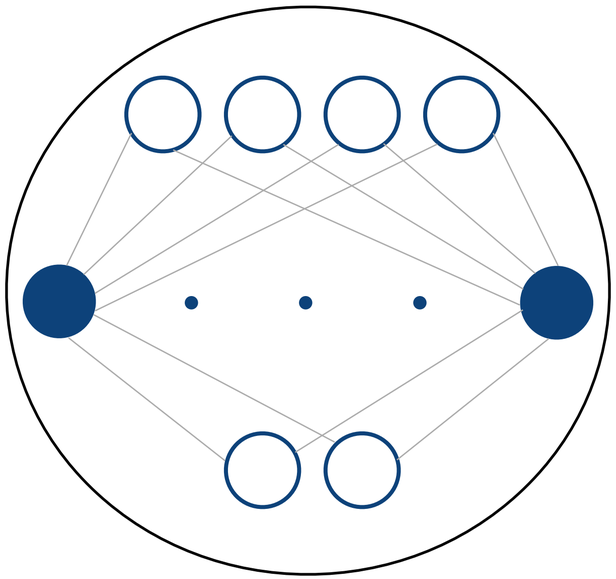
```
x1 = relu([ W1 ] * [ in ] + [ b1 ])
out = linear([ W2 ] * [ x1 ] + [ b2 ])
```

Models are trained in High-Precision

high-precision



input data → training



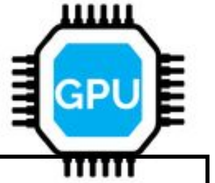
```
x1 = relu([ W1 ]
out = linear([ W2
```

```
[ -5.23724322e-03 ... 1.30853499e-04
                  ...
 -7.29779880e-01 ... -2.27958648e-04 ]
```

64-bit floating-point

Models are trained in High-Precision

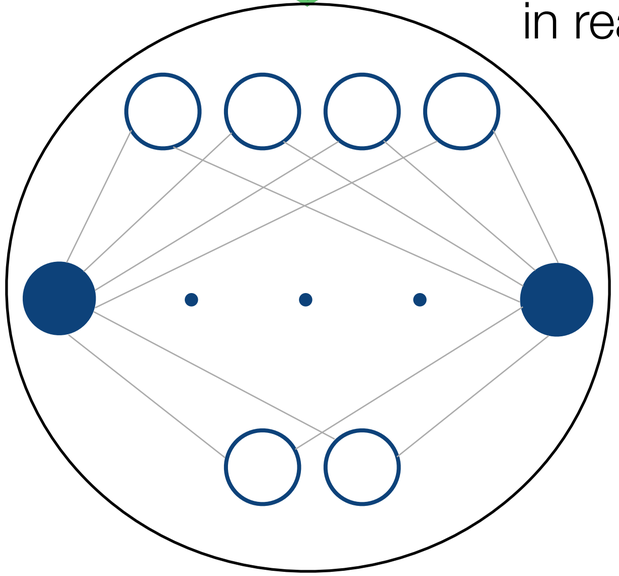
high-precision



input data → training

+/- error

in real-valued arithmetic

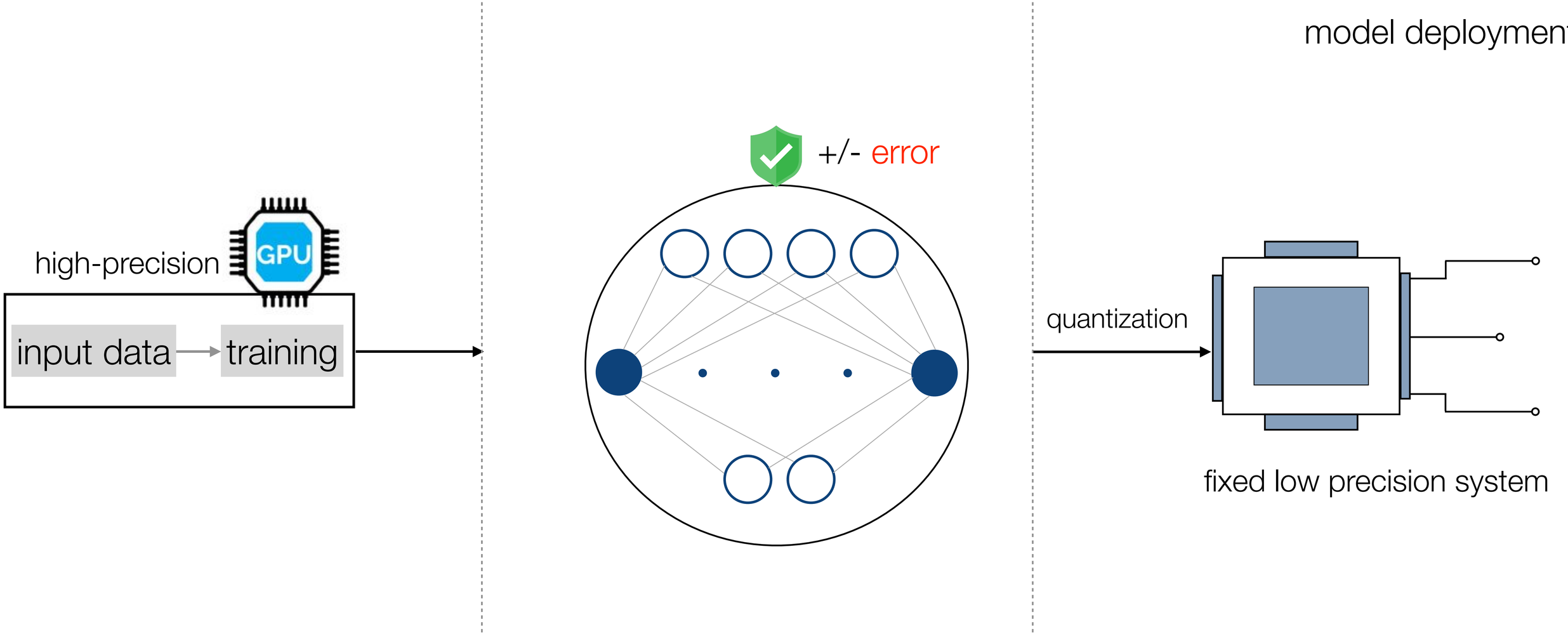


```
x1 = relu([ W1 ]
out = linear([ W2
```

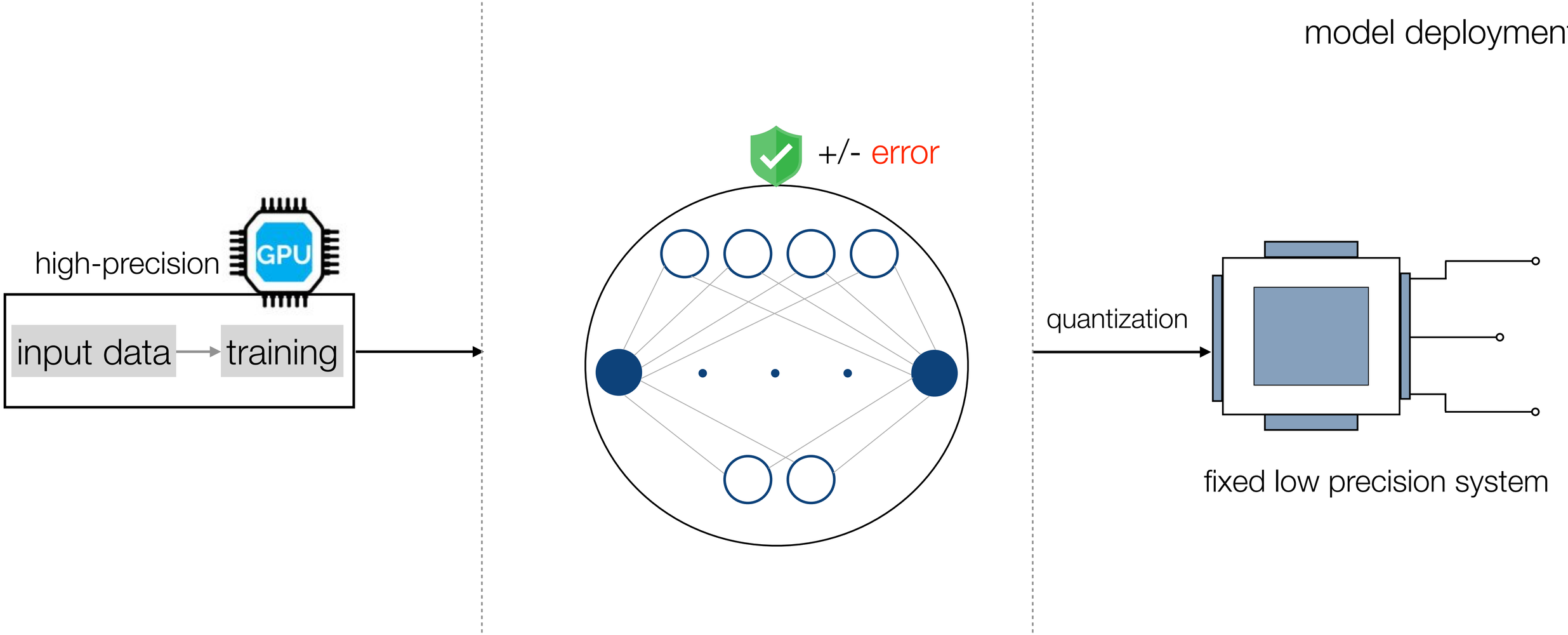
```
[-5.23724322e-03 ... 1.30853499e-04
                 ...
-7.29779880e-01 ... -2.27958648e-04]
```

64-bit floating-point

Model Deployment requires Quantization



Model Deployment requires Quantization

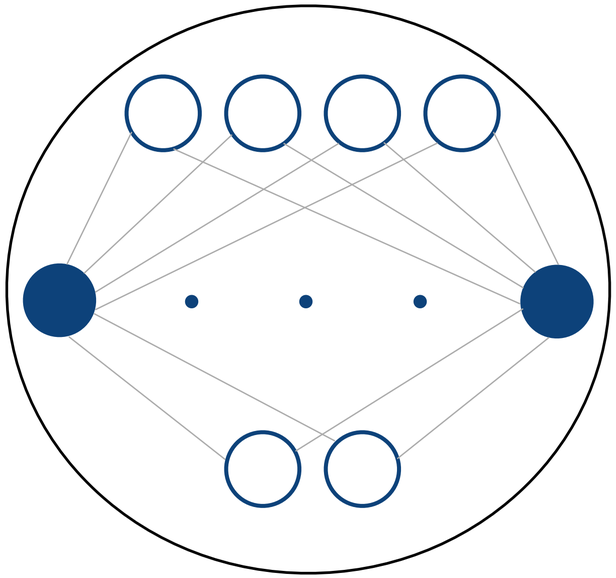


We need to quantize respecting the error bound!

Sound Mixed Fixed-Point Quantization

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
```

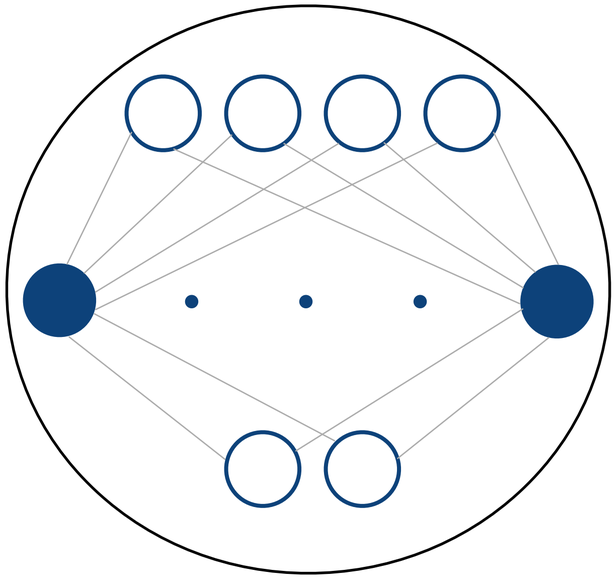

mixed precision
fixed-point code

directly synthesized

XILINX

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
```

no fixed-point support!

FPTuner

Daisy

- not scalable
- needs unrolled structures
- over-approximates a lot

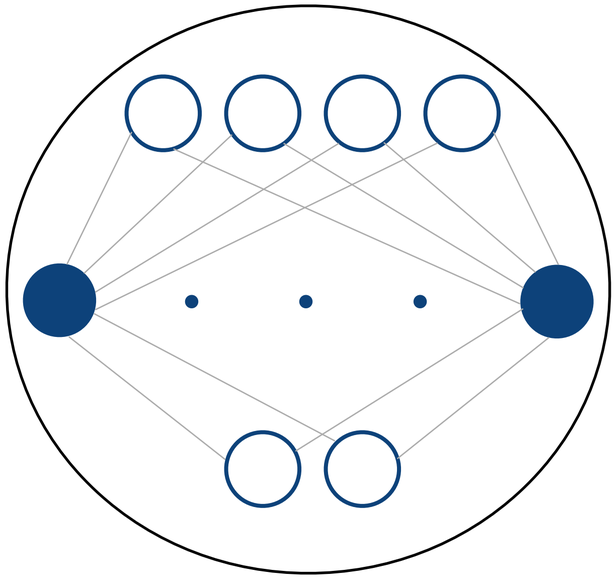
mixed precision
fixed-point code

directly synthesized

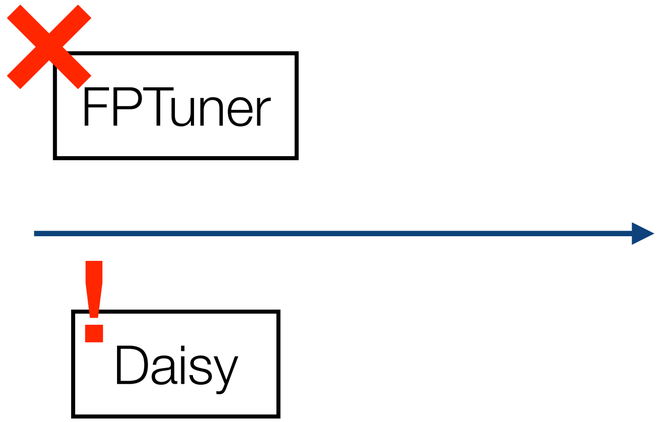
XILINX

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
```

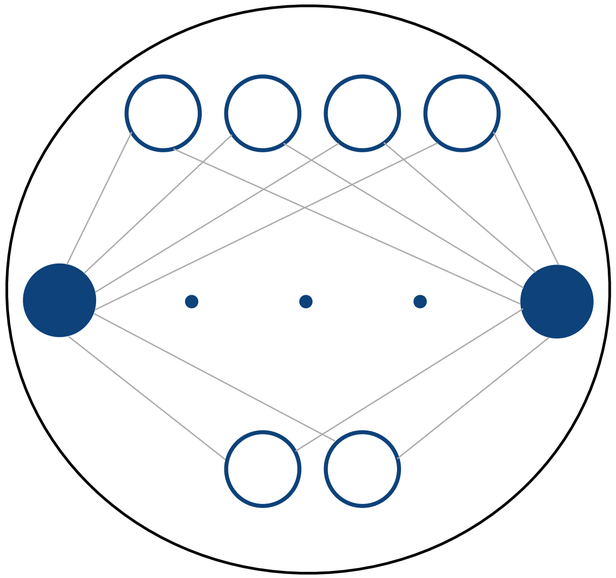


mixed precision
fixed-point code

Our Contribution: Sound Scalable Quantizer for NNs

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
```

```
minimize: precision
```

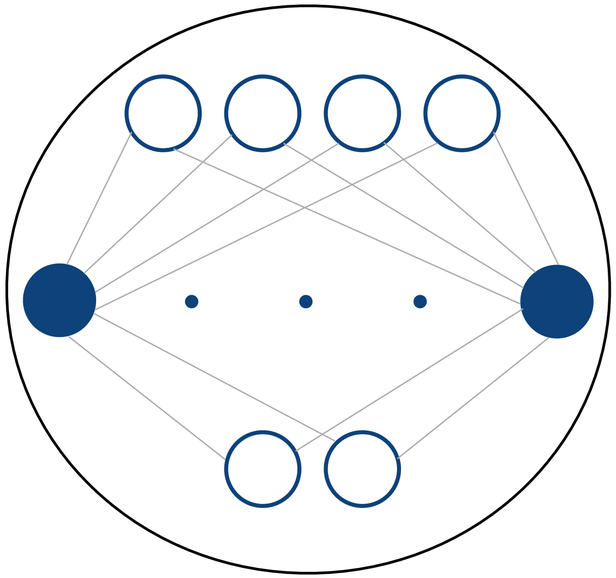
cost function

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

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

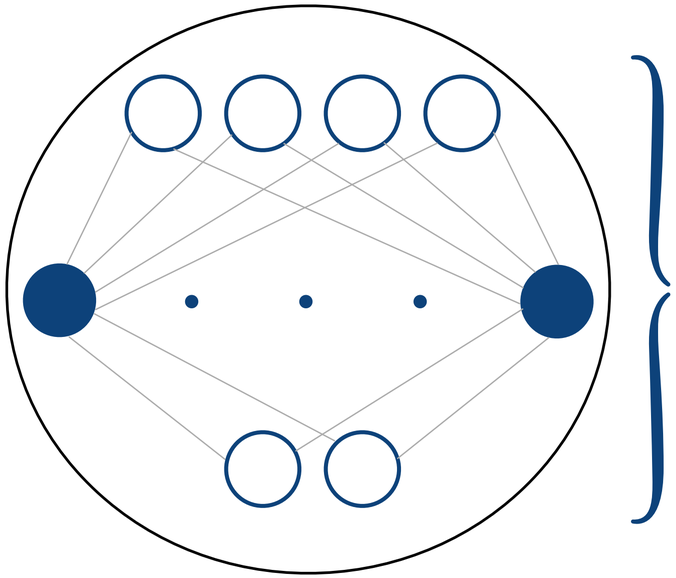
subject to:

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

- integer-valued cost
- real-valued error constraint

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
```



ensure: no overflow

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

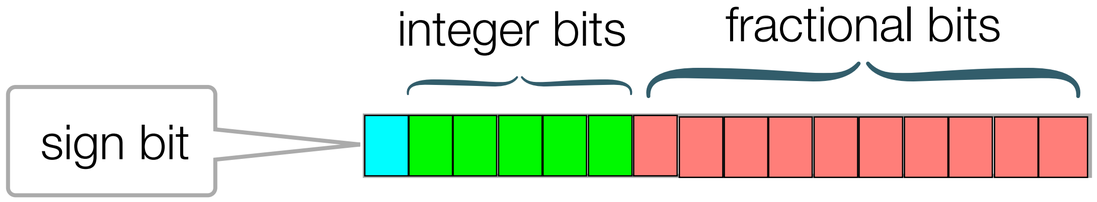
minimize: $\gamma = \sum_{i=1}^{n} \gamma_i^{dot} + \gamma_i^{bias} + \gamma_i^{\alpha}$

subject to:

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

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

- integer-valued cost
- real-valued error constraint
- integer-valued range constraint



Sound Mixed Fixed-Point Quantization

mixed-integer problem

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

subject to:

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

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

Sound Mixed Fixed-Point Quantization

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!

Sound Mixed Fixed-Point Quantization

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 Solution: Reduce to Mixed Integer Linear Programming (MILP) Problem!

Aster: Sound Quantizer for NNs

```
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: Sound Quantizer for NNs

```
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

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 synthesized

XILINX

Summary of Results: Mixed Fixed-Point Quantization of NNs

target error: `1e-3`, max precision: `32`-bit, TO: 5 hours

#benchmarks	#params	analysis time	
		Daisy	Aster
mid-sized (14)	60 - 3920		
large (4)	12K - 44.5K		

Summary of Results: Mixed Fixed-Point Quantization of NNs

target error: `1e-3`, max precision: `32`-bit, TO: 5 hours

#benchmarks	#params	analysis time	
		Daisy	Aster
mid-sized (14)	60 - 3920	4s - 2h 46m 20s	**2s - 50s**
large (4)	12K - 44.5K	TO	**12m 7s - 3h 49m 31s**

Summary of Results: Mixed Fixed-Point Quantization of NNs

target error: `1e-3`, max precision: `32`-bit, TO: 5 hours

#benchmarks	#params	analysis time		latency (clock-cycles)	
		Daisy	Aster	Daisy	Aster
mid-sized (14)	60 - 3920	4s - 2h 46m 20s	**2s - 50s**	12 - 178	**12 - 27**
large (4)	12K - 44.5K	TO	**12m 7s - 3h 49m 31s**	TO	**8K - 13K**

Summary of Results: Mixed Fixed-Point Quantization of NNs

target error: `1e-3`, max precision: `32`-bit, TO: 5 hours

#benchmarks	#params	analysis time		latency (clock-cycles)	
		Daisy	Aster	Daisy	Aster
mid-sized (14)	60 - 3920	4s - 2h 46m 20s	**2s - 50s**	12 - 178	**12 - 27**
large (4)	12K - 44.5K	TO	**12m 7s - 3h 49m 31s**	TO	**8K - 13K**

Aster is more precise than Daisy —
Daisy reports 5 infeasibility, Aster reports **3**!

Takeaways

- Specializing optimization in application contexts can be beneficial
- Optimization with linearizations and abstractions is effective for NNs
- An automated NN quantizer: Aster  
  - generates sound quantized code that can be directly synthesized in Xilinx
  - is precise and scalable

Expanding the Horizons of Finite-Precision Analysis

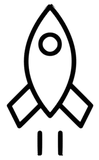
Accuracy Analysis

Optimization

Thesis Contributions

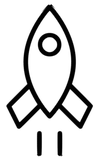
iFM '19 EMSOFT '18

Probabilistic Analysis



TACAS '21

Static + Dynamic Analysis



EMSOFT '23

NN Quantization

worst-case error analysis for small programs

Daisy FLUCTUAT Rosa

FPTaylor PRECiSA ...

worst-case tuning for small (floating-point) programs

Daisy FPTuner

Future Research Directions

- Scalable Accuracy Analysis
  - considering probabilistic inputs
  - by combining static, dynamic analysis and machine learning techniques

Future Research Directions

- Scalable Accuracy Analysis
  - considering probabilistic inputs
  - by combining static, dynamic analysis and machine learning techniques
- Scalable Optimization
  - considering probabilistic inputs
  - specialize in other application contexts

Future Research Directions

- Scalable Accuracy Analysis
  - considering probabilistic inputs
  - by combining static, dynamic analysis and machine learning techniques
- Scalable Optimization
  - considering probabilistic inputs
  - specialize in other application contexts
- Finite-precision in the context of
  - heterogeneous HPC systems

… and others!

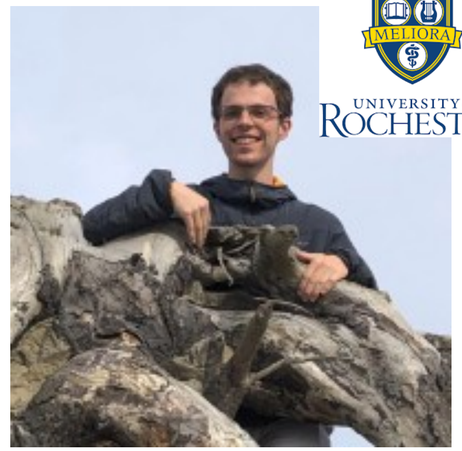
Collaborators

Eva Darulova

Sylvie Putot

Eric Goubault

Milos Prokop



Joshua Sobel

Clothilde Jeangoudoux

Maria Christakis

Anastasia Volkova

Thank You
For Your Attention!