Reasoning about Floating Point in Real-World Systems

Wonyeol Lee (Stanford CS)

PhD Oral Exam (05/15/2023)

“Continuous” Computations

Continuous values

$$6,\ 2.5,\ \frac{3}{7},\ \sqrt{2},\ 0.9\pi,\ \ldots$$

Operations on them

$$6+2.5,\ \frac{3}{7}\times\sqrt{2},\ \cos(0.9\pi),\ \ldots$$

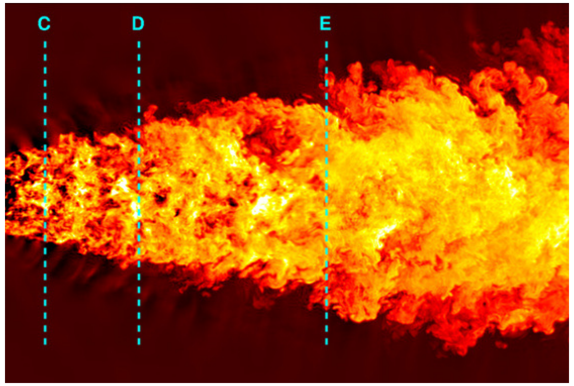
“Continuous” Computations

Continuous values

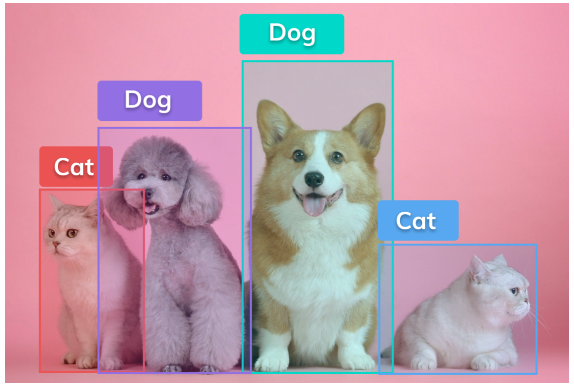
$$6,\ 2.5,\ \frac{3}{7},\ \sqrt{2},\ 0.9\pi,\ \ldots$$

Operations on them

$$6+2.5,\ \frac{3}{7}\times\sqrt{2},\ \cos(0.9\pi),\ \ldots$$



Scientific computing



Machine learning

Computer graphics

…

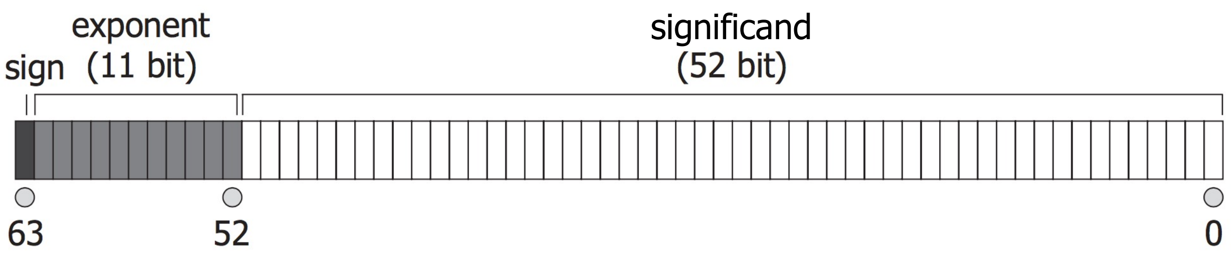
Theory and Practice

	Continuous values	Operations on them
Theory	Real numbers ($\mathbb{R}$)	Exact operations ($+$, $\times$, ...)

Theory and Practice

	Continuous values	Operations on them
Theory	Real numbers ($\mathbb{R}$)	Exact operations ($+, \times, \ldots$)
Practice	Floating-point numbers ($\mathbb{F}$)	Floating-point operations ($+_{\mathbb{F}}, \times_{\mathbb{F}}, \ldots$)

e.g., 64-bit double-precision floats



$$\underbrace{1}\ \underbrace{01111111111}\ \underbrace{1100\cdots00}_{(2)}$$

$$= (-1)^{1} \cdot 2^{1023-1023} \cdot 1.110\cdots00_{(2)}$$

Discrepancy

uncountably many　　　　exact

	Continuous values	Operations on them
Theory	Real numbers ($\mathbb{R}$)	Exact operations ($+$, $\times$, ...)
Practice	Floating-point numbers ($\mathbb{F}$)	Floating-point operations ($+_{\mathbb{F}}$, $\times_{\mathbb{F}}$, ...)

finitely many　　　　inexact

Discrepancy

uncountably many   exact

	Continuous values	Operations on them
Theory	Real numbers ($\mathbb{R}$)	Exact operations ($+, \times, \ldots$)
Practice	Floating-point numbers ($\mathbb{F}$)	Floating-point operations ($+_{\mathbb{F}}, \times_{\mathbb{F}}, \ldots$)

finitely many   inexact

- Discrepancy between the theory and practice of "continuous" computations.
- Can we better understand/characterize this discrepancy arising in real-world systems?

My Work

Programs that implement `math.h`.	Correctness: [PLDI'16], [POPL'18].
Programs that train ML models.	Acceleration: [Submitted].
Programs that compute derivatives.	Correctness: [ICML'23].

- Discrepancy between the theory and practice of "continuous" computations.
- Can we better understand/characterize this discrepancy arising in real-world systems?

My Work

done during my 3-year leave from Stanford

Programs that implement `math.h`.	Correctness: [PLDI'16], [POPL'18].
Probabilistic / differentiable programming.	Correctness: [NeurIPS'18/20], [POPL'20/23].
Programs that train ML models.	Acceleration: [Submitted].
Programs that compute derivatives.	Correctness: [ICML'23].

- Discrepancy between the theory and practice of "continuous" computations.
- Can we better understand/characterize this discrepancy arising in real-world systems?

My Work

Programs that implement `math.h`.	Correctness: [PLDI'16], [POPL'18].
Probabilistic / differentiable programming.	Correctness: [NeurIPS'18/20], [POPL'20/23].
Programs that train ML models.	Acceleration: [Submitted].
Programs that compute derivatives.	Correctness: [ICML'23].

`math.h` Implementations

$$\log x$$

mathematical specification $\boldsymbol{f}$

$\boldsymbol{X} = (0, 2^{1024})$

input interval $\boldsymbol{X}$

```
<log>
...
subsd  %xmm5, %xmm1
mulpd  %xmm0, %xmm5
...
psllq  $0xc,  %xmm1
pshufd $0xe4, %xmm5, %xmm6
...
```

`math.h` implementation $\boldsymbol{P}$

`math.h` Implementations

infinite # bits

$\log x$

mathematical specification $\boldsymbol{f}$

∃ precision loss

$\boldsymbol{X} = (0, 2^{1024})$

input interval $\boldsymbol{X}$

fixed # bits

```
<log>
...
subsd  %xmm5, %xmm1
mulpd  %xmm0, %xmm5
...
psllq  $0xc,  %xmm1
pshufd $0xe4, %xmm5, %xmm6
...
```

`math.h` implementation $\boldsymbol{P}$

Problem

$$\log x$$

mathematical specification $\boldsymbol{f}$

$\exists$ precision loss

$\longleftrightarrow$

$\boldsymbol{X} = (0, 2^{1024})$

input interval $\boldsymbol{X}$

```
<log>
...
subsd  %xmm5, %xmm1
mulpd  %xmm0, %xmm5
...
psllq  $0xc,  %xmm1
pshufd $0xe4, %xmm5, %xmm6
...
```

`math.h` implementation $\boldsymbol{P}$

- Can we find a tight bound on the maximum precision loss automatically?

Problem

$$\log x$$

mathematical specification $\boldsymbol{f}$

∃ precision loss

$\longleftrightarrow$

$\boldsymbol{X} = (0, 2^{1024})$

input interval $\boldsymbol{X}$

```
<log>
...
subsd  %xmm5, %xmm1
mulpd  %xmm0, %xmm5
...
psllq  $0xc,  %xmm1
pshufd $0xe4, %xmm5, %xmm6
...
```

`math.h` implementation $\boldsymbol{P}$

- Can we find a tight bound on the maximum precision loss automatically?
- Goal: Find a small $\Theta > 0$ in an automatic way such that

$$\mathrm{err}\big(\boldsymbol{f}(x), \boldsymbol{P}(x)\big) \le \Theta \quad \text{for all } x \in \boldsymbol{X}.$$

Two Challenges

$$\log x$$

mathematical specification $\boldsymbol{f}$

```
<log>
...
subsd  %xmm5, %xmm1
mulpd  %xmm0, %xmm5
...
psllq  $0xc,  %xmm1
pshufd $0xe4, %xmm5, %xmm6
...
```

`math.h` implementation $\boldsymbol{P}$

(1) $\boldsymbol{P}$ often mixes floating-point and bit-level operations. [PLDI'16]

Two Challenges

$$\log x$$

mathematical specification $\boldsymbol{f}$

```
<log>
...
subsd  %xmm5, %xmm1
mulpd  %xmm0, %xmm5
...
psllq  $0xc,  %xmm1
pshufd $0xe4, %xmm5, %xmm6
...
```

`math.h` implementation $\boldsymbol{P}$

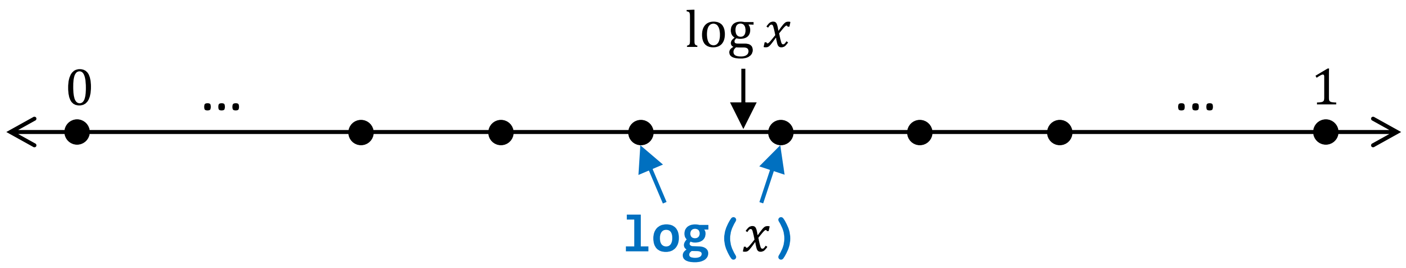
(1) $\boldsymbol{P}$ often mixes floating-point and bit-level operations. [PLDI'16]

(2) $\boldsymbol{P}$ is often claimed to have a very small precision loss. [POPL'18]

e.g., < 1 ulp

Two Challenges

- `log` has precision loss of < 1 ulp $\iff$ for any $x \in \boldsymbol{X}$,



- 0.5 ulp is the best we can achieve (by definition).

(1) $\boldsymbol{P}$ often mixes floating-point and bit-level operations. [PLDI'16]

(2) $\boldsymbol{P}$ is often claimed to have a very small precision loss. [POPL'18]

e.g., < 1 ulp

Two Challenges

Prior work on the problem [FM’15, POPL’14, FMICS’09, PLDI’03, FMCAD’00, …]:

- requires considerable manual efforts; or
- cannot handle general mixed codes and prove small error bounds.

(1) $\boldsymbol{P}$ often mixes floating-point and bit-level operations. [PLDI’16]

(2) $\boldsymbol{P}$ is often claimed to have a very small precision loss. [POPL’18]

Two Challenges

Prior work on the problem [FM'15, POPL'14, FMICS'09, PLDI'03, FMCAD'00, …]:

- requires considerable manual efforts; or
- cannot handle general mixed codes and prove small error bounds.

(1) $\boldsymbol{P}$ often mixes floating-point and bit-level operations. [PLDI'16]

(2) $\boldsymbol{P}$ is often claimed to have a very small precision loss. [POPL'18]

Bit-Level Operations

- Example: Given n (in `int`), compute 2^n (in `double`).
  - Naïve solution: `int_to_double(1 << n)`. Slow, correct for $n \in [0, 31]$.

Bit-Level Operations

- Example: Given n (in `int`), compute 2^n (in `double`).
  - Naïve solution: `int_to_double(1 << n)`. Slow, correct for $n \in [0, 31]$.
  - Better solution: `(n + 1023) << 52`. Fast, correct for $n \in [-1022, 1023]$.

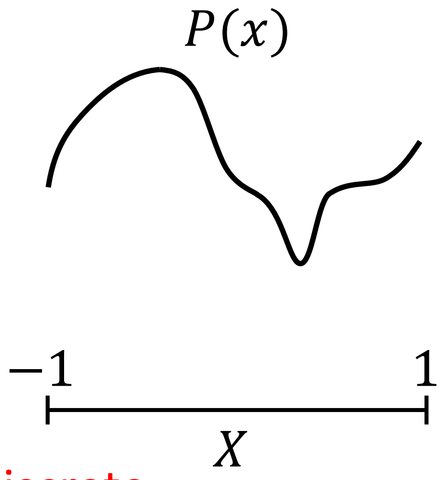
Bit-Level Operations

- Example: Given n (in `int`), compute 2^n (in `double`).
  - Naïve solution: `int_to_double(1 << n)`. Slow, correct for $n \in [0, 31]$.
  - Better solution: `(n + 1023) << 52`. Fast, correct for $n \in [-1022, 1023]$.

- Such operations are often used in highly optimized implementations of `math.h`.

It is difficult to reason about such "mixed codes", which intermix bit-level and floating-point operations.

Bit-Level Operations

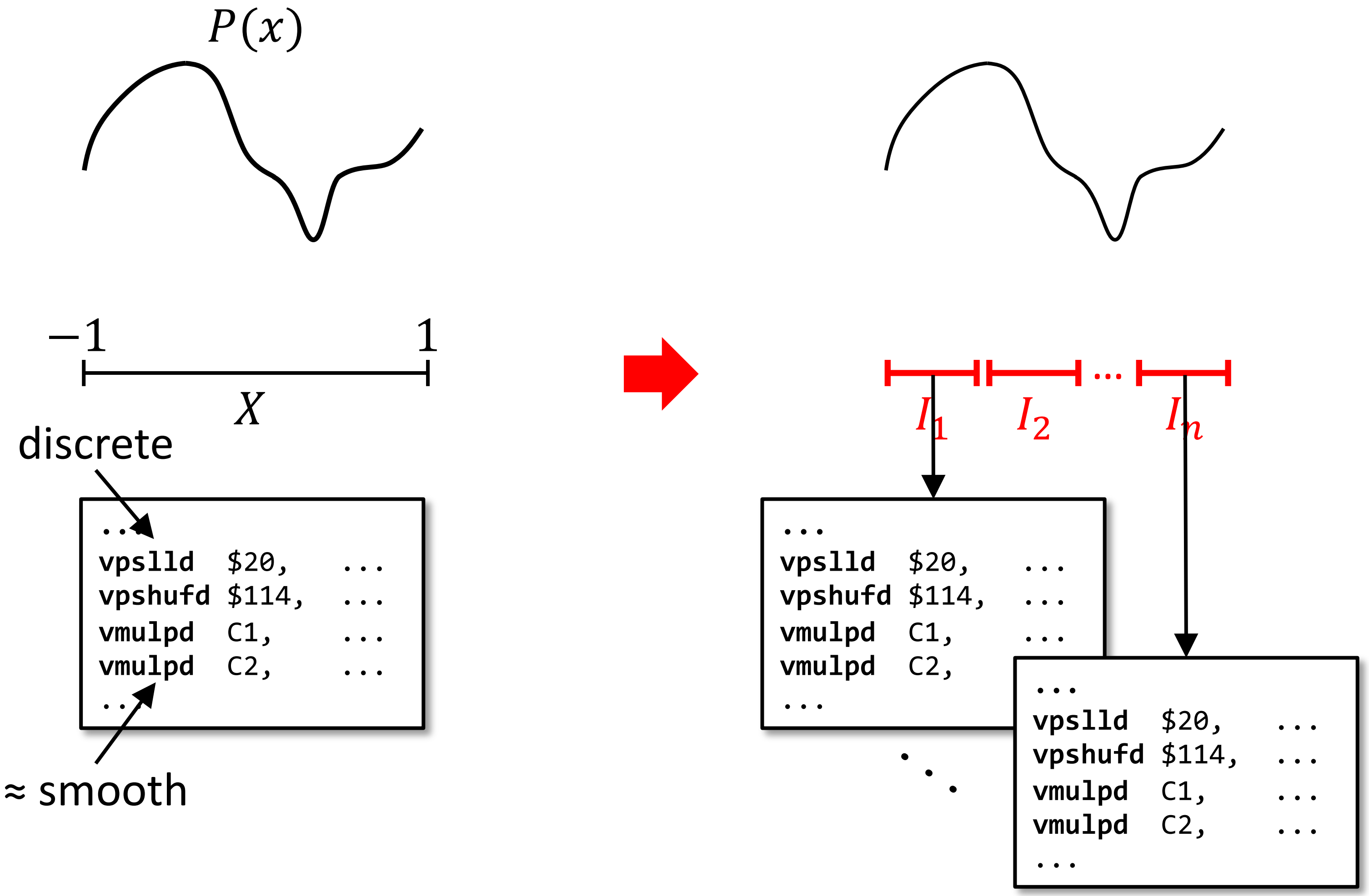


discrete

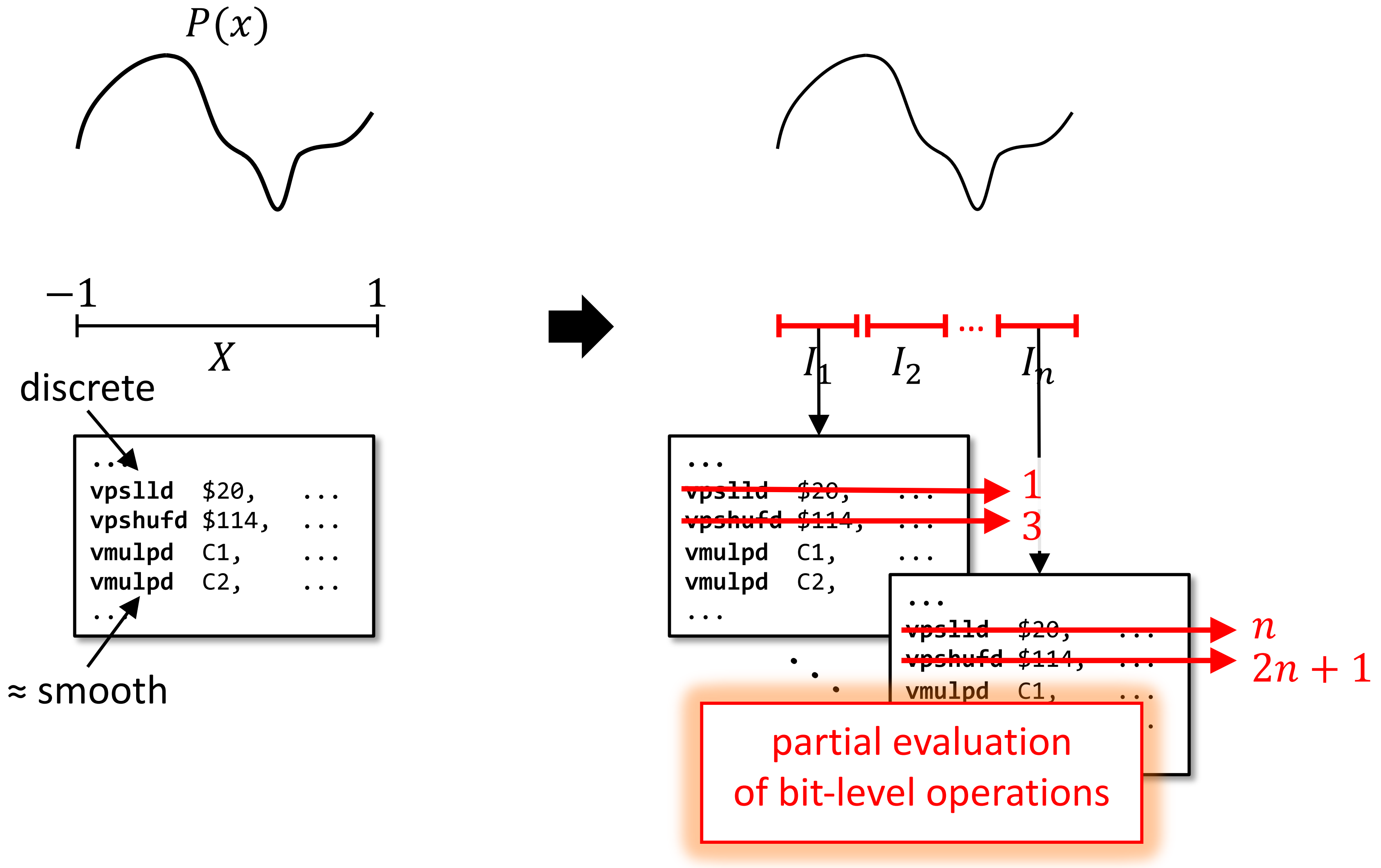
```
...
vpslld  $20,    ...
vpshufd $114,   ...
vmulpd  C1,     ...
vmulpd  C2,     ...
..
```

≈ smooth

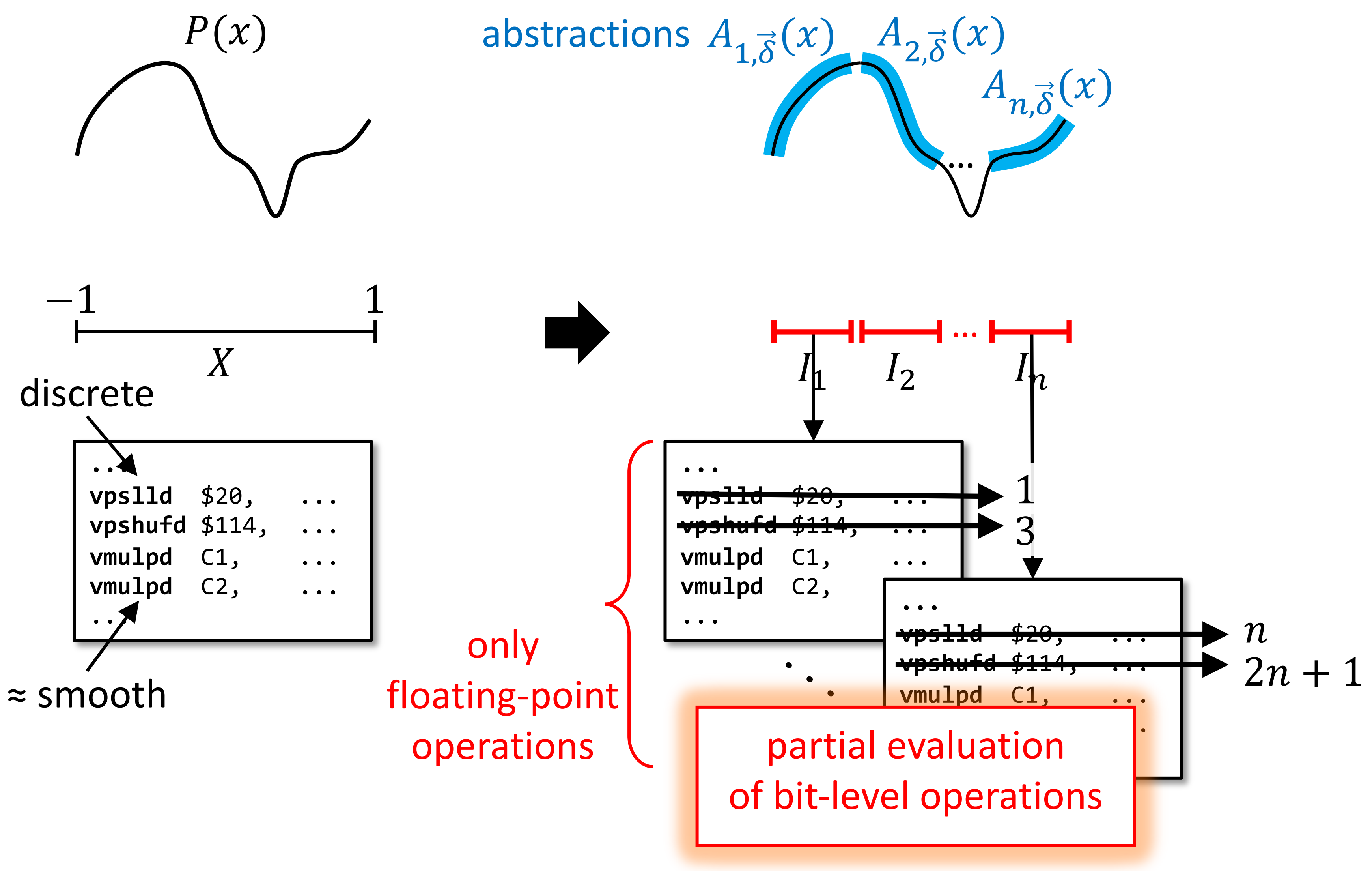
Our Approach



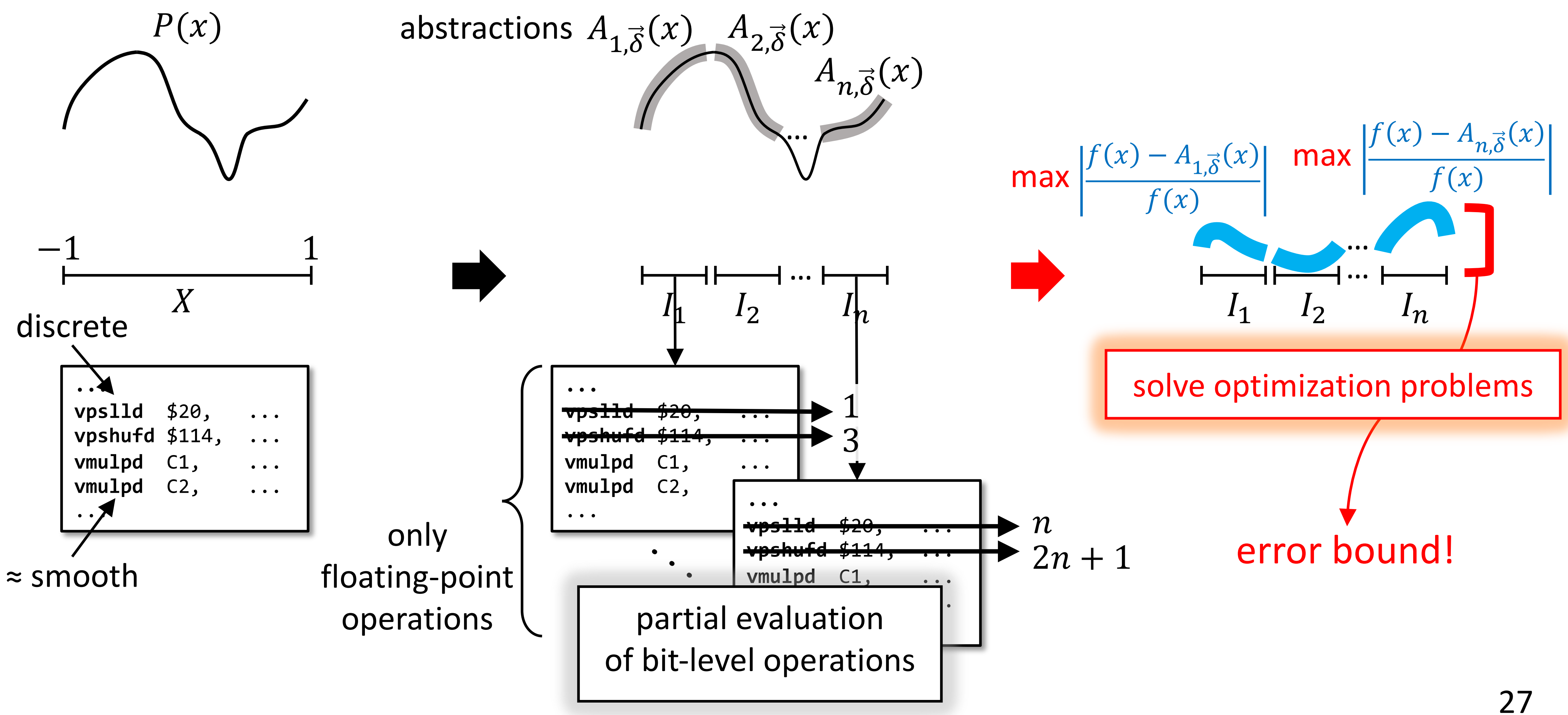
Our Approach



Our Approach



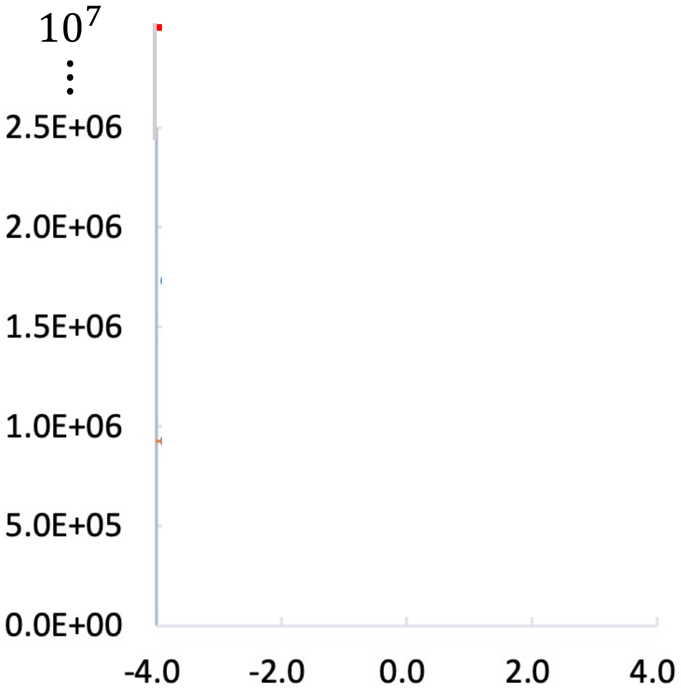
Our Approach

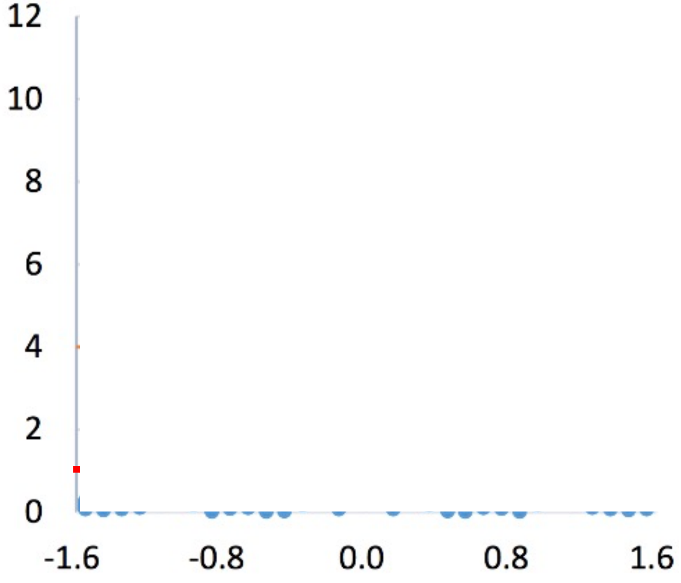


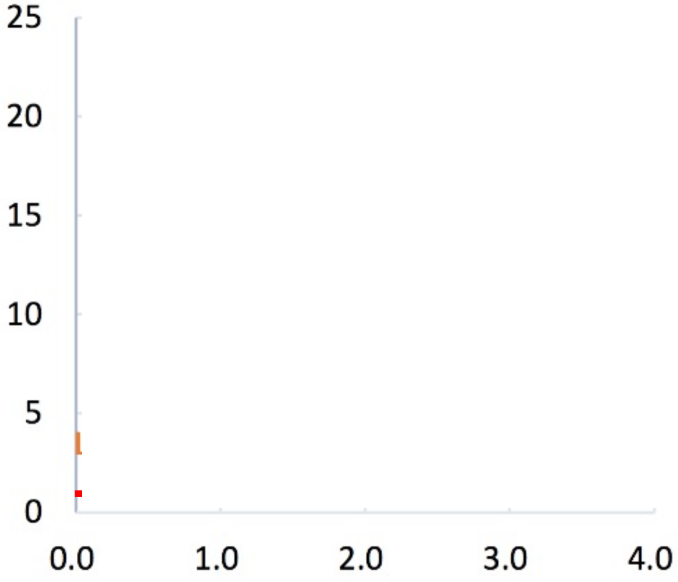
Evaluation Results

Optimized implementation	Intel's implementation of `math.h`	
$\mathtt{exp}_{\mathtt{opt}}$	`sin`	`log`

error (ulp)





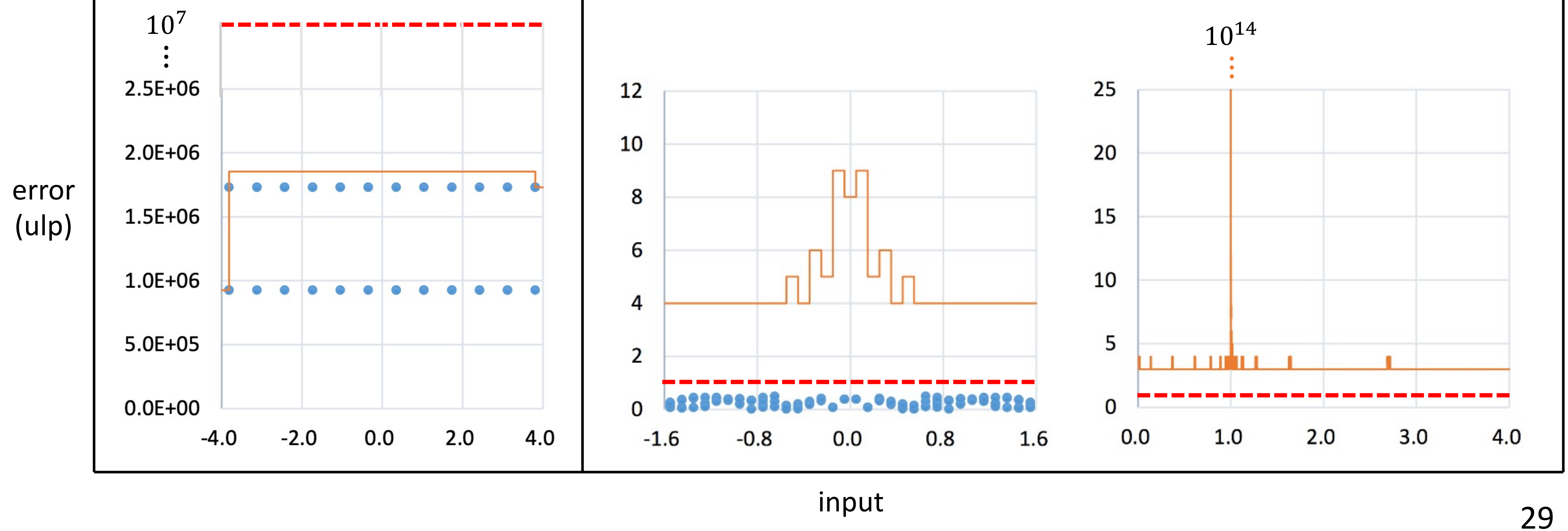


input

Evaluation Results

Our error bounds [PLDI'16]
Claimed error bounds
Actual errors (between intervals)

Optimized implementation	Intel's implementation of `math.h`	
$\texttt{exp}_{\texttt{opt}}$	`sin`	`log`



Summary of Contributions [PLDI'16]

- We propose the first systematic, automatic method for verifying mixed codes.
- Our method is based on abstraction, analytic optimization, and testing.

  Key: Split the input range into sub-intervals so that bit-level op's can be partially evaluated.
- We apply our method to real-world binaries for `math.h` and prove their formal error bonds.

Two Challenges

$\log x$ ⟷

mathematical specification $\boldsymbol{f}$

```
<log>
...
subsd  %xmm5, %xmm1
mulpd  %xmm0, %xmm5
...
psllq  $0xc,  %xmm1
pshufd $0xe4, %xmm5, %xmm6
...
```

`math.h` implementation $\boldsymbol{P}$

(1) $\boldsymbol{P}$ often mixes floating-point and bit-level operations. [PLDI'16]

(2) $\boldsymbol{P}$ is often claimed to have a very small precision loss. [POPL'18]

e.g., < 1 ulp

Exactness Properties

- Floating-point operations are often inexact.

$$a \times_{\mathbb{F}} 2^n \neq a \times 2^n$$

$$a -_{\mathbb{F}} b \neq a - b$$

Exactness Properties

- Floating-point operations are often inexact, but sometimes exact.

$$a \times_{\mathbb{F}} 2^n = a \times 2^n \quad \text{if } |a \times 2^n| \geq 2^{-1022}. \qquad \text{[Folklore]}$$

$$a -_{\mathbb{F}} b = a - b \quad \text{if } b/2 \leq a \leq 2b. \qquad \text{[Sterbenz, 1973]}$$

Exactness Properties

- Floating-point operations are often inexact, but sometimes exact.

$$a \times_{\mathbb{F}} 2^n = a \times 2^n \qquad \text{if } |a \times 2^n| \geq 2^{-1022}. \qquad \text{[Folklore]}$$

$$a -_{\mathbb{F}} b = a - b \qquad \text{if } b/2 \leq a \leq 2b. \qquad \text{[Sterbenz, 1973]}$$

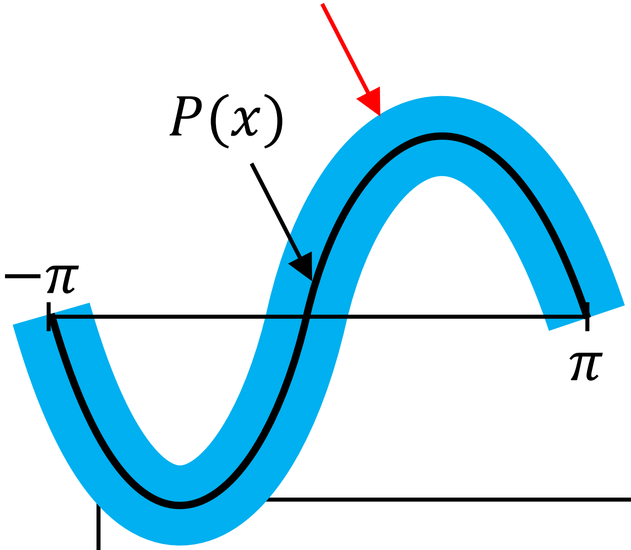
$$[\cdots] \qquad \text{if} \qquad [\cdots]$$

- Such properties are implicitly used in highly accurate implementations of `math.h`.

Standard error analysis techniques ignore these exactness properties.

Loose Error Bounds

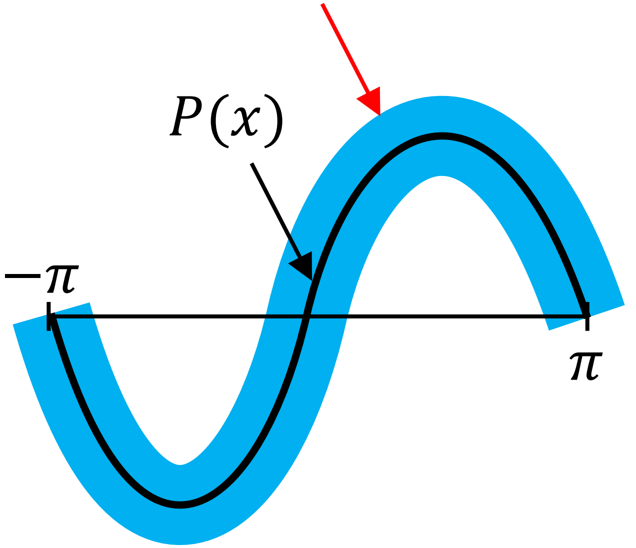
This sometimes results in
too overapproximate abstractions.



Standard error analysis techniques ignore these exactness properties.

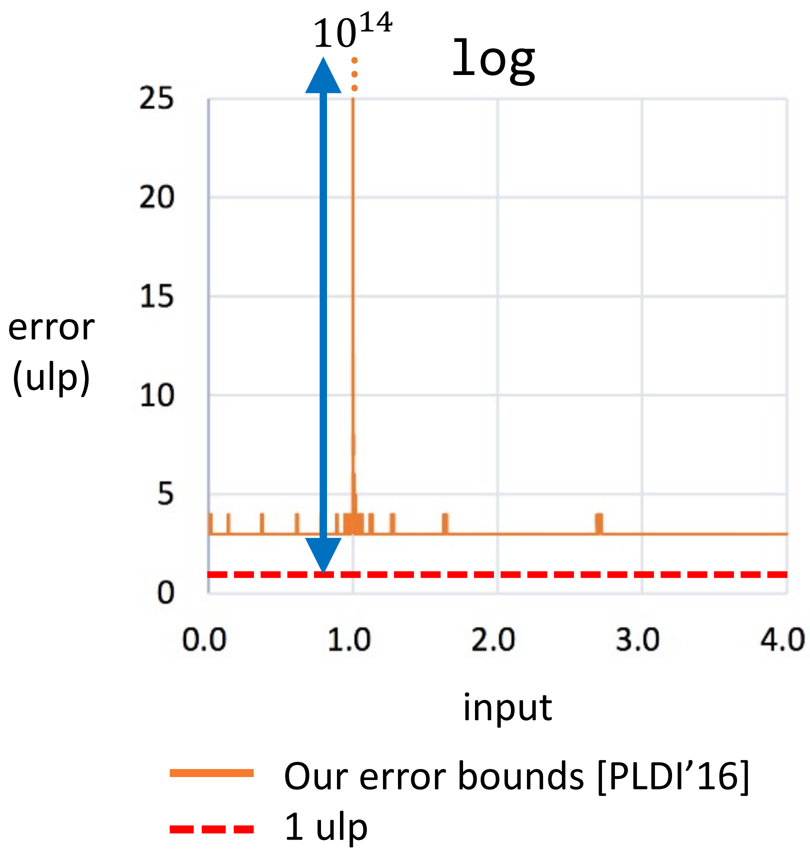
Loose Error Bounds

This sometimes results in
too overapproximate abstractions.



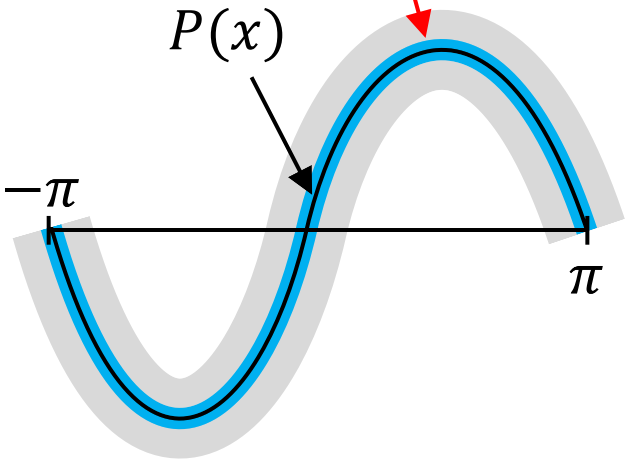
$\Rightarrow$

Intel's implementation of `math.h`



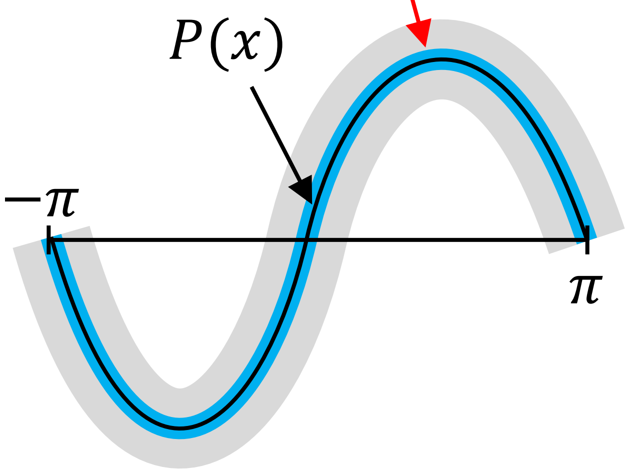
Our Approach

Construct tighter abstractions by automatically applying exactness results.



Our Approach

Construct tighter abstractions by automatically applying exactness results. ➡ Check preconditions of exactness results.

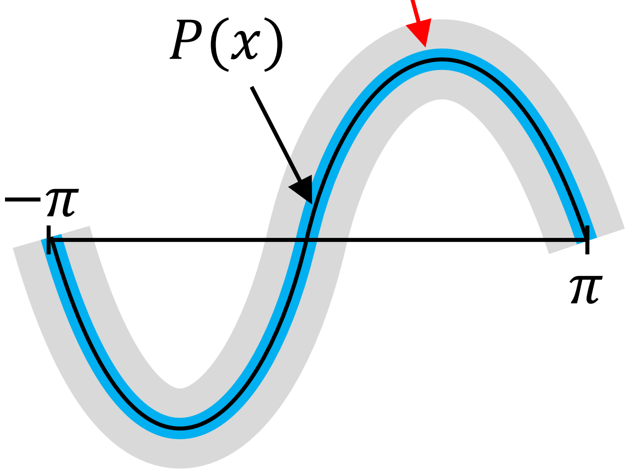


Example: Can we apply "$e(x) -_{\mathbb{F}} e'(x) = e(x) - e'(x)$"?

- Need: ${}^{1}/_{2}\, e(x) \leq e'(x) \leq 2e(x)$ for all $x \in I$.

Our Approach

Construct tighter abstractions by automatically applying exactness results.



Check preconditions of exactness results.

Solve optimization problems.

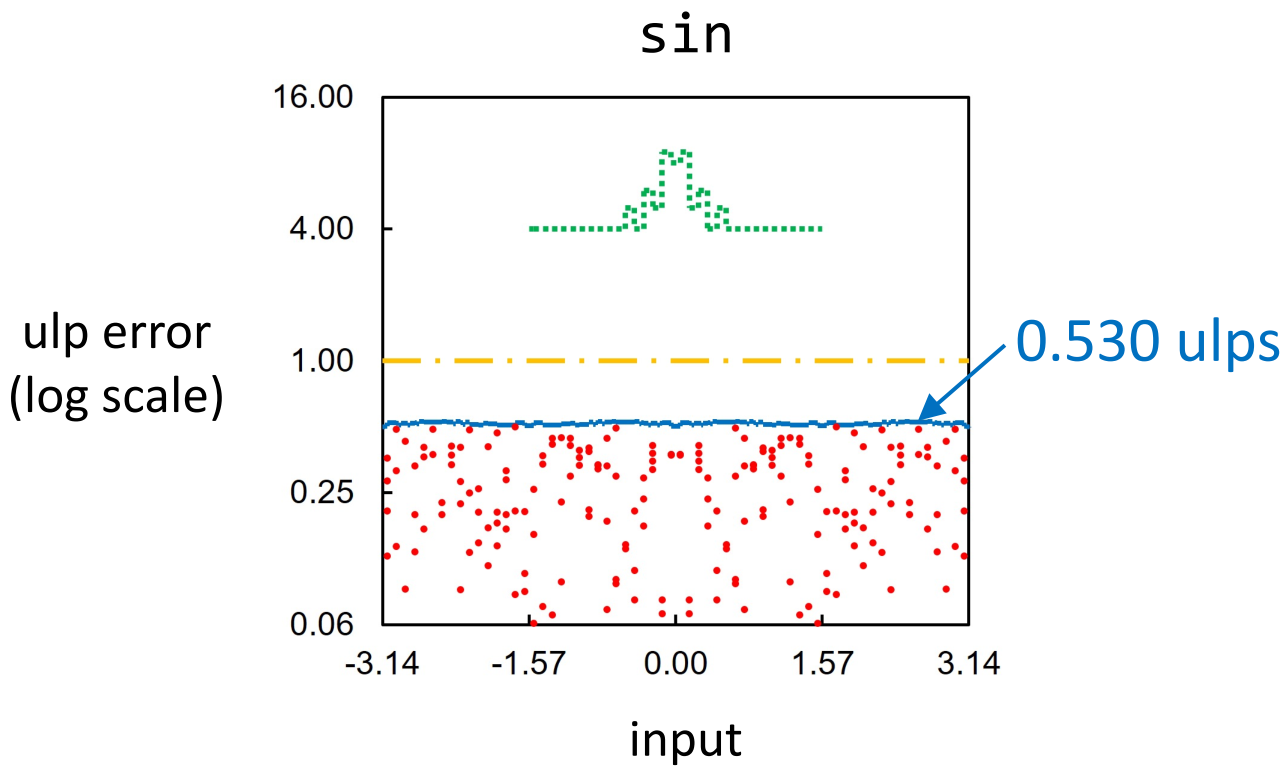
Example: Can we apply "$e(x) -_{\mathbb{F}} e'(x) = e(x) - e'(x)$"?

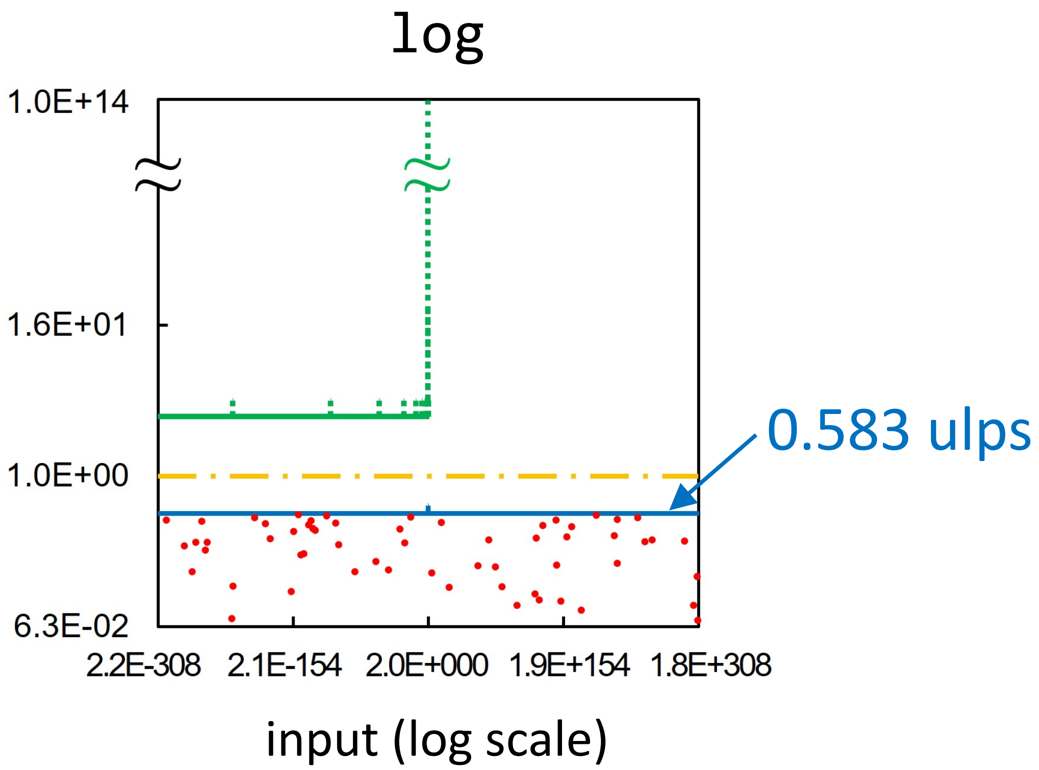
- Need: ${}^1\!/_2\, e(x) \le e'(x) \le 2e(x)$ for all $x \in I$.
- Check: $\min_{x \in I}\left(e'(x) - {}^1\!/_2\, e(x)\right) \ge 0$
  $\land \max_{x \in I}\left(e'(x) - 2e(x)\right) \le 0.$

Evaluation Results

error bounds from [PLDI'16]
error bounds from [POPL'18]
1 ulp
actual ulp errors

Intel's implementation of `math.h`





Summary of Contributions [POPL'18]

- We propose the first automatic method for verifying `math.h` implementations.
- Our method is based on a reduction of this verification problem to optimization problems.
  Key: Apply exactness results, after checking their preconditions by solving opt'n problems.
- We apply our method to Intel's `math.h` implementations and prove their correctness.

Summary of Contributions [POPL’18]

- We propose the first automatic method for verifying `math.h` implementations.

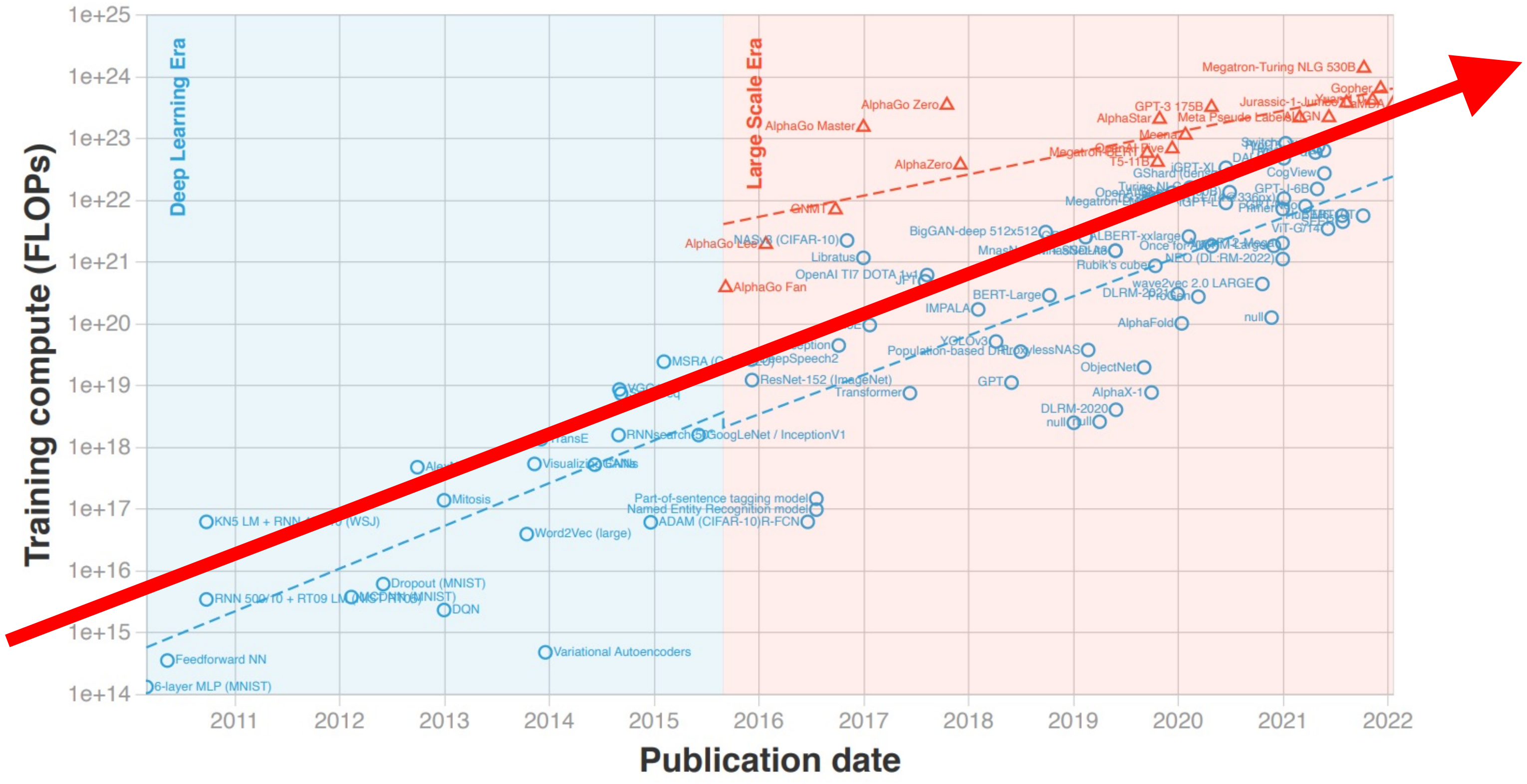
- Our method is based on a reduction of this verification problem to optimization problems.
  Key: Apply exactness results, after checking their preconditions by solving opt’n problems.

- We apply our method to Intel’s `math.h` implementations and prove their correctness.

Agenda

Programs that implement `math.h`.	Correctness: [PLDI'16], [POPL'18].
Probabilistic / differentiable programming.	Correctness: [NeurIPS'18/20], [POPL'20/23].
Programs that train ML models.	Acceleration: [Submitted].
Programs that compute derivatives.	Correctness: [ICML'23].

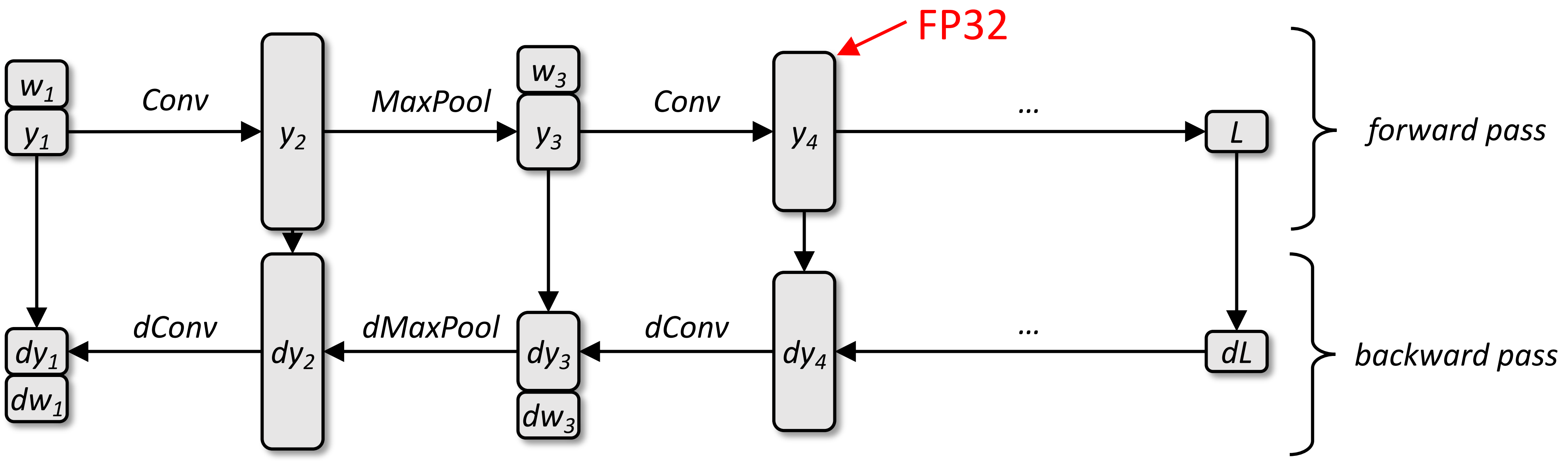
Training in Machine Learning



How to accelerate training computation while maintaining training quality?

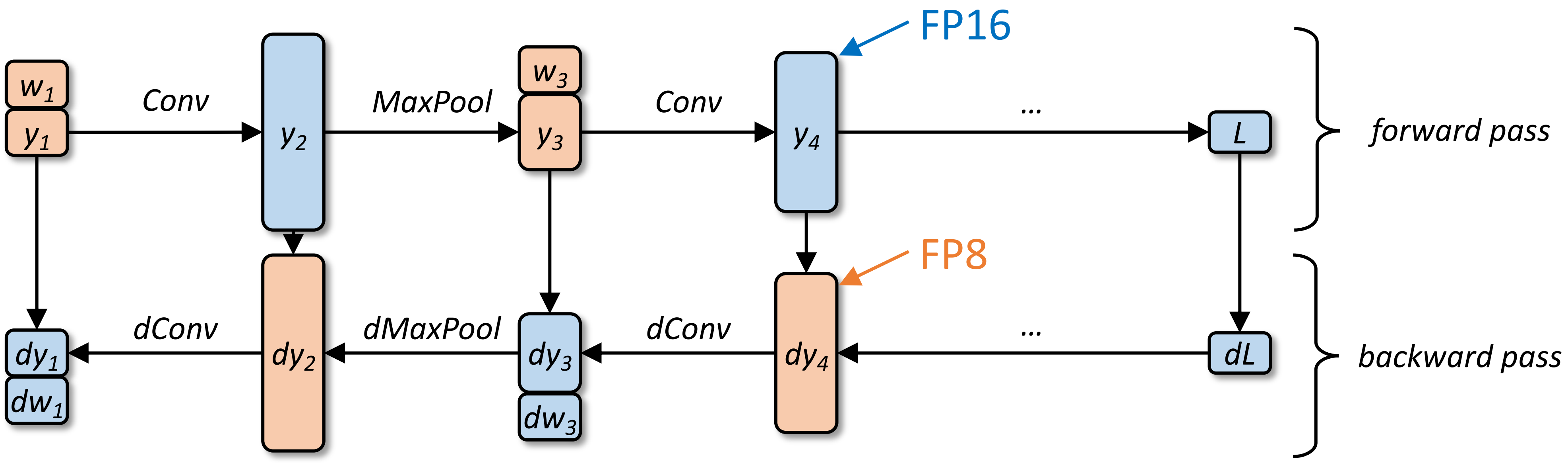
Source: “Compute trends across three eras of machine learning”, IJCNN’22.

Low-Precision Training



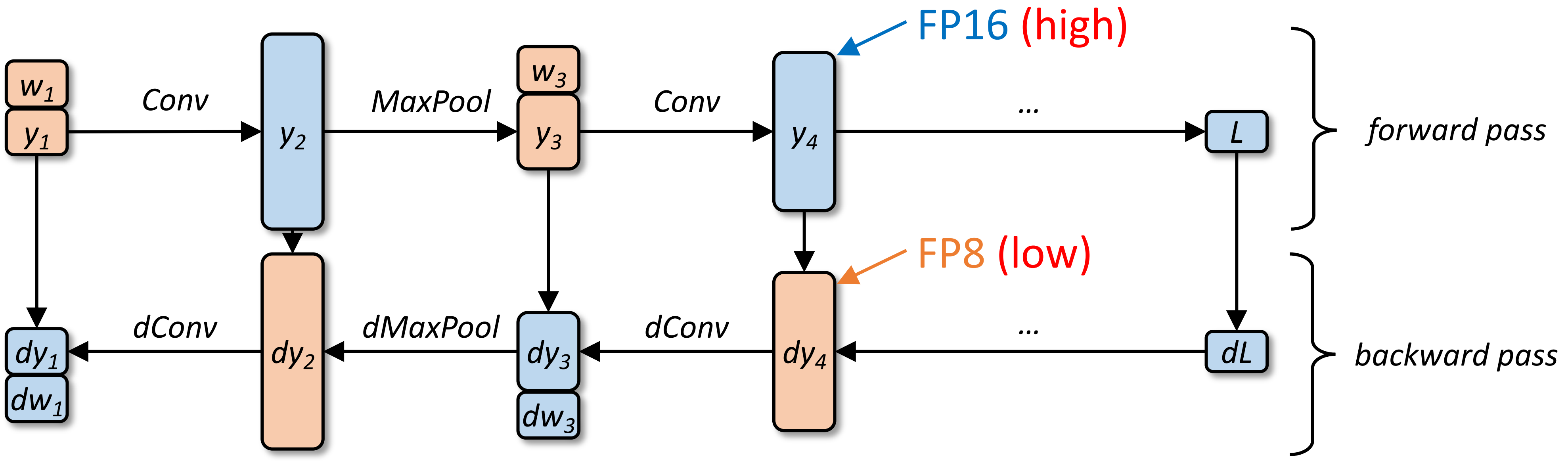
- Standard training: Use FP32 to represent tensors.

Low-Precision Training



- Standard training: Use FP32 to represent tensors.
- Low-precision training: Use <FP32 to represent tensors.

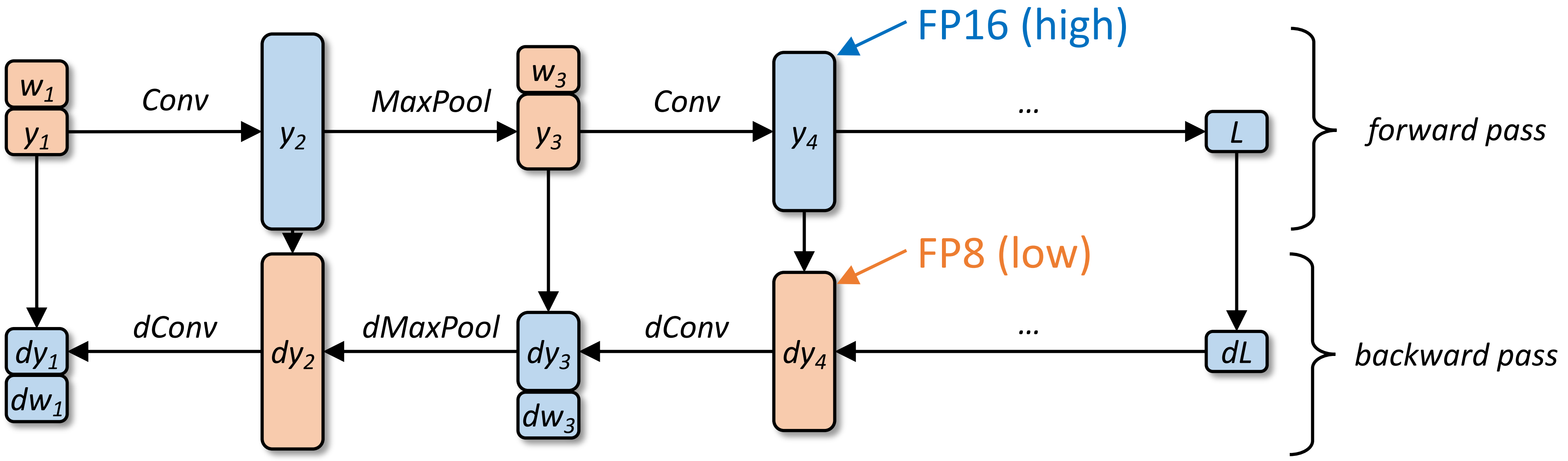
Low-Precision Training



- Standard training: Use FP32 to represent tensors.
- Low-precision training: Use <FP32 to represent tensors.
  - Consider and apply two precision levels: high and low.
  - We call a mapping from tensors to {high, low} a "precision assignment".

Low-Precision Training

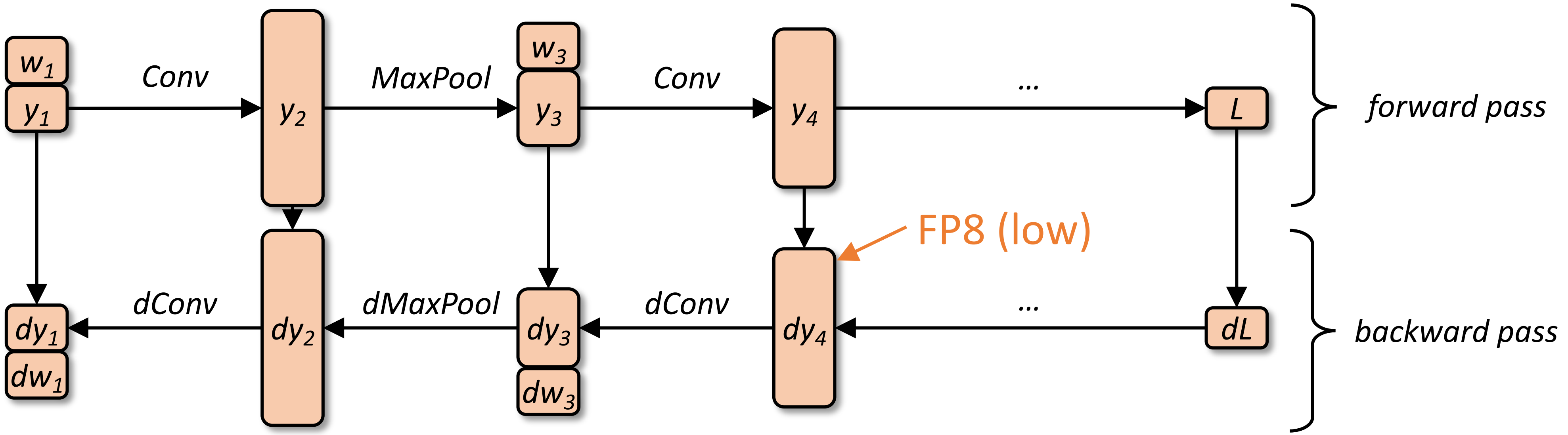
$\pi_{\text{op}} =$



- Standard training: Use FP32 to represent tensors.
- Low-precision training: Use <FP32 to represent tensors.
  - Consider and apply two precision levels: high and low.
  - We call a mapping from tensors to {high, low} a "precision assignment".
  - Example: operator-based assignment π_{op}.

Low-Precision Training

$\pi_{\text{unif}} =$



- Standard training: Use FP32 to represent tensors.
- Low-precision training: Use <FP32 to represent tensors.
  - Consider and apply two precision levels: high and low.
  - We call a mapping from tensors to {high, low} a "precision assignment".
  - Example: operator-based assignment π_{op}, uniform assignment π_{unif}.

Limitations

most often just one

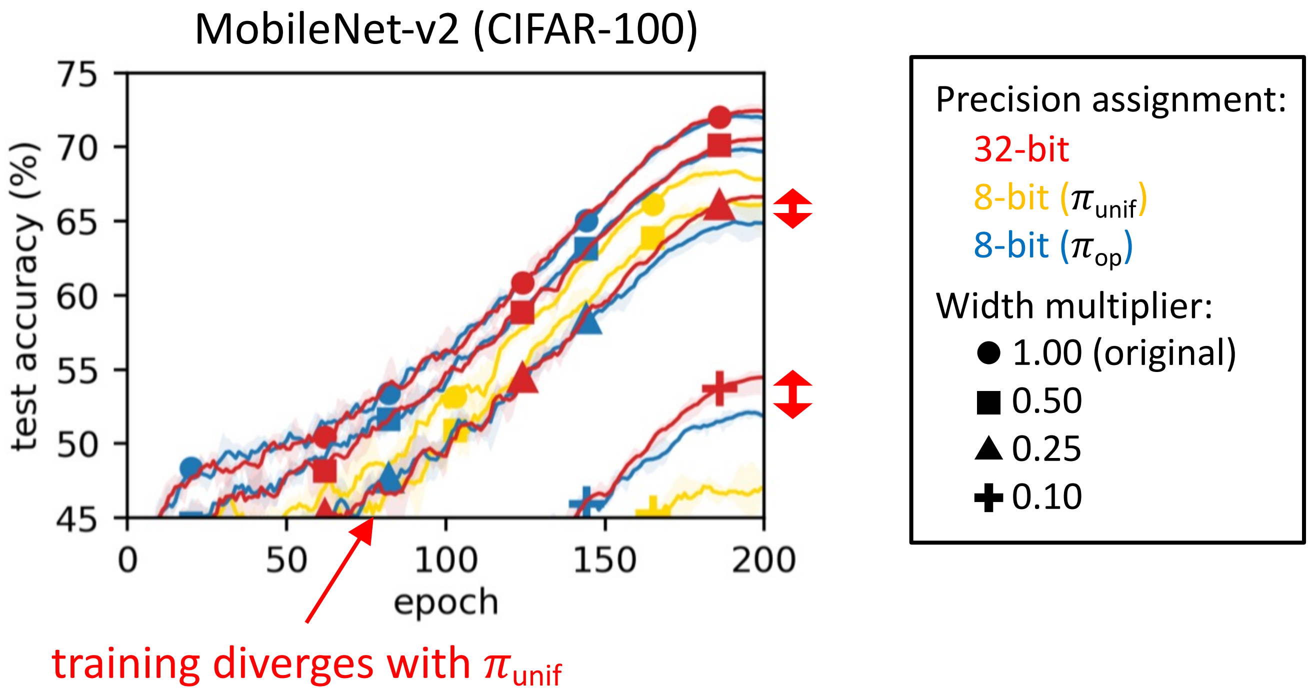
- For a given set of models, prior work uses very few precision assignments (e.g., π_{unif} or π_{op}).

Limitations

- For a given set of models, prior work uses very few precision assignments (e.g., π_{unif} or π_{op}).
- But for other models, the chosen π
  - may result in noticeably worse accuracy (and divergence of training).
  - may admit more efficient assignments that achieve similar accuracy.

Limitations

- For a given set of models, prior work uses very few precision assignments (e.g., π_{unif} or π_{op}).
- But for other models, the chosen π
  - may result in noticeably worse accuracy (and divergence of training).
  - may admit more efficient assignments that achieve similar accuracy.



Memory-Accuracy Tradeoff Problem

- Input: a model M, FP formats $(FP_{\mathrm{hi}}, FP_{\mathrm{lo}})$, and a parameter $r \in [0,1]$.
- Goal: Find a precision assignment π for M using only $(FP_{\mathrm{hi}}, FP_{\mathrm{lo}})$ such that

Memory-Accuracy Tradeoff Problem

- Input: a model M, FP formats $(FP_{\mathrm{hi}}, FP_{\mathrm{lo}})$, and a parameter $r \in [0,1]$.
- Goal: Find a precision assignment π for M using only $(FP_{\mathrm{hi}}, FP_{\mathrm{lo}})$ such that

$\mathrm{acc}_M(\pi)$ is maximized subject to $\mathrm{ratio}_{\mathrm{lo}}(\pi) \geq r$.

accuracy of M when trained with π

"low-precision ratio" of π

$=$ ratio of FP_{lo}-tensors in π

$\approx$ reduction in memory/time for training

E.g., $\mathrm{ratio}_{\mathrm{lo}}(\pi_{\mathrm{hi}}) = 0$, $\mathrm{ratio}_{\mathrm{lo}}(\pi_{\mathrm{lo}}) = 1$.

Memory-Accuracy Tradeoff Problem

- Input: a model M, FP formats $(FP_{\mathrm{hi}}, FP_{\mathrm{lo}})$, and a parameter $r \in [0,1]$.
- Goal: Find a precision assignment π for M using only $(FP_{\mathrm{hi}}, FP_{\mathrm{lo}})$ such that

$\mathrm{acc}_M(\pi)$ is maximized subject to $\mathrm{ratio}_{\mathrm{lo}}(\pi) \geq r$.

accuracy of M when trained with π

"low-precision ratio" of π
$=$ ratio of FP_{lo}-tensors in π
$\approx$ reduction in memory/time for training

- Find π that maximizes training accuracy under a memory/time constraint (given by r).

Challenges

- Input: a model M, FP formats $(FP_{\text{hi}}, FP_{\text{lo}})$, and a parameter $r \in [0,1]$.
- Goal: Find a precision assignment π for M using only $(FP_{\text{hi}}, FP_{\text{lo}})$ such that

$$\text{acc}_M(\pi) \text{ is maximized} \qquad \text{subject to} \qquad \text{ratio}_{\text{lo}}(\pi) \geq r.$$

- Practically,
  - There is no known analytic method for predicting $\text{acc}_M(\pi)$.
  - There are exponentially many candidates for π.

Challenges

- Input: a model M, FP formats $(FP_{\mathrm{hi}}, FP_{\mathrm{lo}})$, and a parameter $r \in [0,1]$.
- Goal: Find a precision assignment π for M using only $(FP_{\mathrm{hi}}, FP_{\mathrm{lo}})$ such that

$$\mathrm{acc}_M(\pi) \text{ is maximized} \qquad \text{subject to} \qquad \mathrm{ratio}_{\mathrm{lo}}(\pi) \geq r.$$

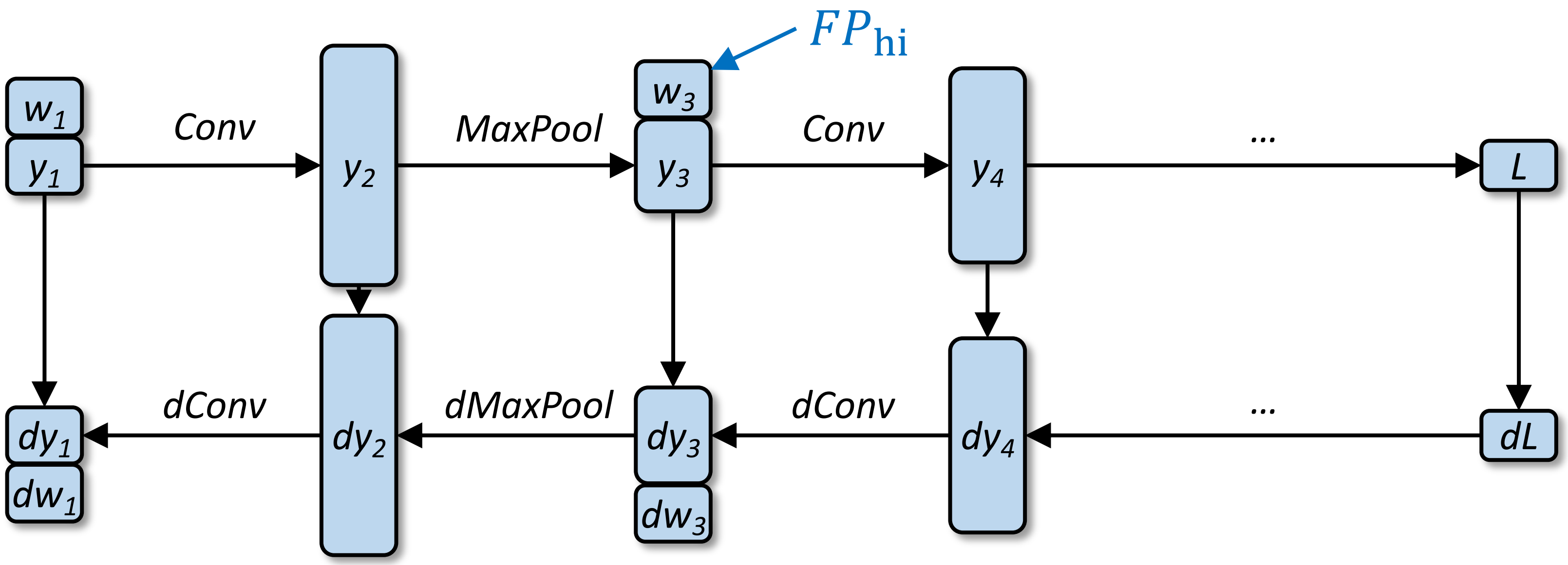
- Practically,
  - There is no known analytic method for predicting $\mathrm{acc}_M(\pi)$.
  - There are exponentially many candidates for π.
- Theoretically, we prove:

  <u>Theorem</u> The memory-accuracy tradeoff problem is NP-hard.

Our Method

- Input: a model M, FP formats $(FP_{\text{hi}}, FP_{\text{lo}})$, and a parameter $r \in [0,1]$.
- Our method for **the tradeoff problem:**
  - Initialize π to the all-FP_{hi} assignment.

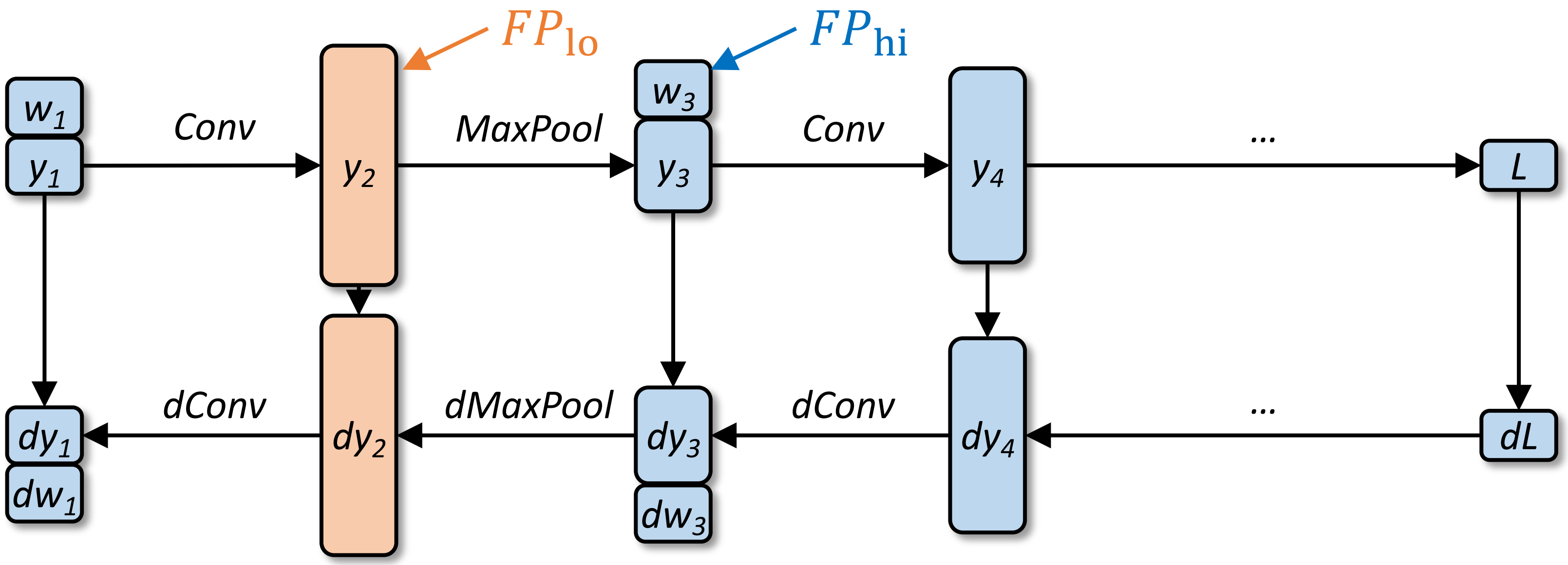
$\pi =$



Our Method

- Input: a model M, FP formats $(FP_{\text{hi}}, FP_{\text{lo}})$, and a parameter $r \in [0,1]$.
- Our method for **the tradeoff problem:**
  - Initialize π to the all-FP_{hi} assignment.
  - Demote the precision of largest tensors (in size) to FP_{lo}.

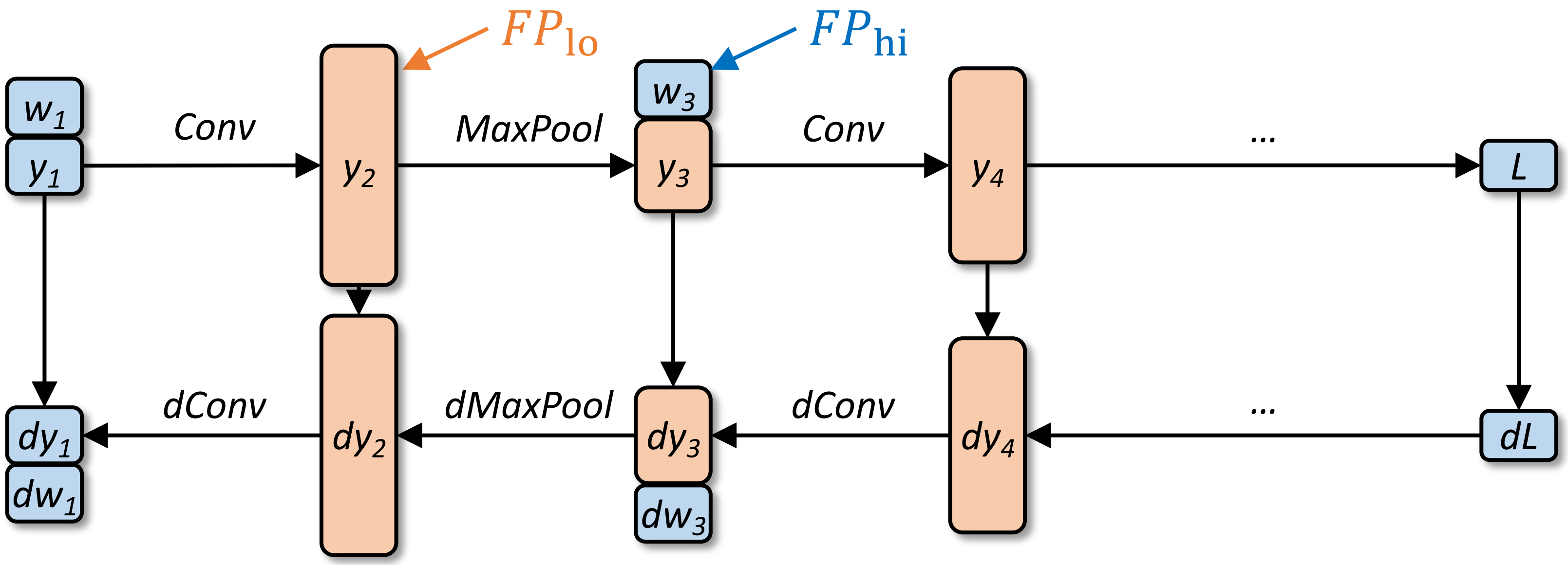
$\pi =$



Our Method

- Input: a model M, FP formats $(FP_{\mathrm{hi}}, FP_{\mathrm{lo}})$, and a parameter $r \in [0,1]$.
- Our method for **the tradeoff problem:**
  - Initialize π to the all-FP_{hi} assignment.
  - Demote the precision of largest tensors (in size) to FP_{lo}.
  - Repeat it while $\mathrm{ratio}_{\mathrm{lo}}(\pi) \geq r$. Return the final π.

$\pi =$

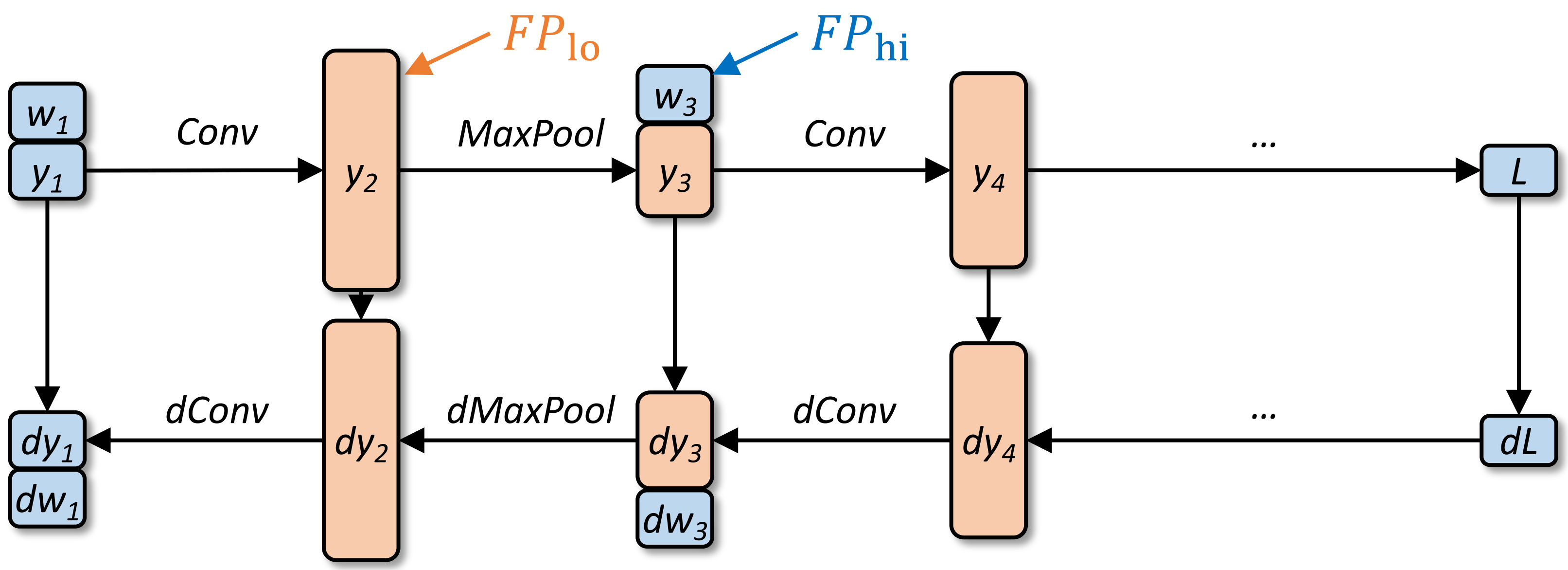


Our Method

- Input: a model M, FP formats $(FP_{\mathrm{hi}}, FP_{\mathrm{lo}})$, and a parameter $r \in [0,1]$.
- Our method for **the tradeoff problem:**
  - Initialize π to the all-FP_{hi} assignment.
  - Demote the precision of largest tensors (in size) to FP_{lo}.
  - Repeat it while $\mathrm{ratio}_{\mathrm{lo}}(\pi) \geq r$. Return the final π.

- Optimal in a very simplified setting.
- Empirically better than other orders.

$\pi =$



Our Method

- Input: a model M, FP formats $(FP_{\mathrm{hi}}, FP_{\mathrm{lo}})$, and a parameter $r \in [0,1]$.
- Our method for **the tradeoff problem:**
  - Initialize π to the all-FP_{hi} assignment.
  - Demote the precision of largest tensors (in size) to FP_{lo}.
  - Repeat it while $\boxed{\mathrm{ratio}_{\mathrm{lo}}(\pi) \geq r}$. Return the final π.
- The above method places no explicit constraint on $\boxed{\mathrm{acc}_M(\pi)}$.
  - Observe: Training with this π sometimes diverges, due to too many overflows.

Our Method

- Input: a model M, FP formats $(FP_{\mathrm{hi}}, FP_{\mathrm{lo}})$, and a parameter $r \in [0,1]$.

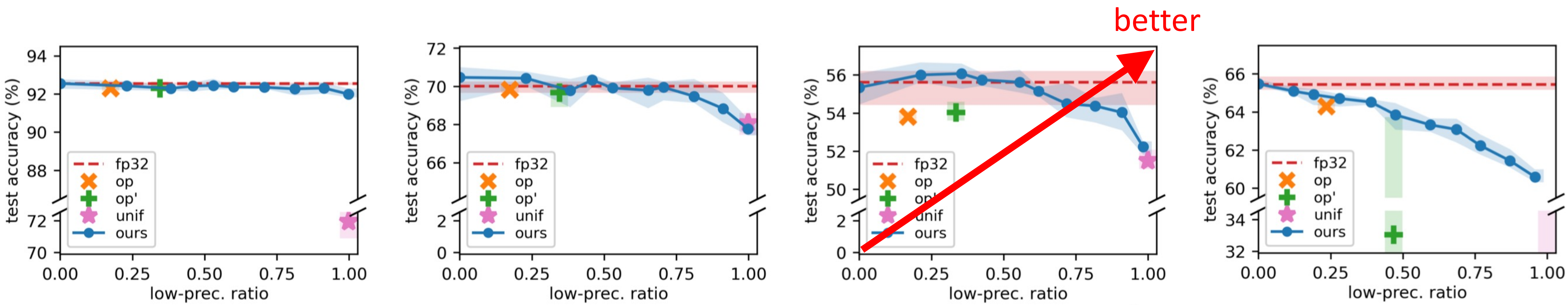
- Our method for **the tradeoff problem:**
    - Initialize π to the all-FP_{hi} assignment.
    - Demote the precision of largest tensors (in size) to FP_{lo}. ⟵ before training
    - Repeat it while $\mathrm{ratio}_{\mathrm{lo}}(\pi) \geq r$. Return the final π.

- The above method places no explicit constraint on $\mathrm{acc}_M(\pi)$.
    - Observe: Training with this π sometimes diverges, due to too many overflows.

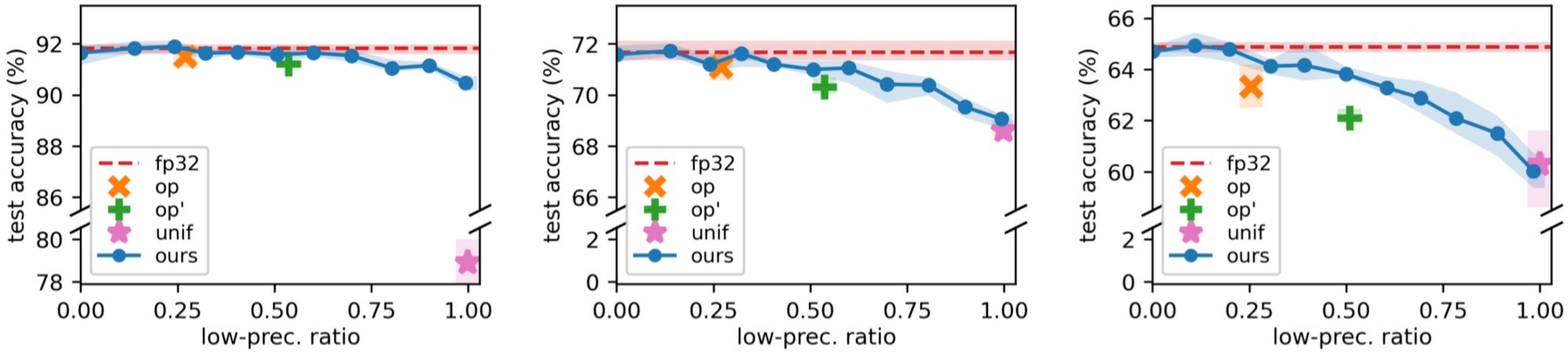
- Our method for **handling overflows:**
    - Promote the precision of tensors that overflow "too much" to FP_{hi}. ⟵ during training

Evaluation Results

- Comparison with existing precision assignments.



(a) CIFAR-10, SqueezeNet (b) CIFAR-100, SqueezeNet (c) CIFAR-100, SqueezeNet† ImageNet, ShuffleNet-v2



(d) CIFAR-10, ShuffleNet-v2 (e) CIFAR-100, ShuffleNet-v2 (f) CIFAR-100, ShuffleNet-v2†

†: width multiplier of 0.25

Summary of Contributions [Submitted]

- We formally introduce the memory-accuracy tradeoff problem and prove its NP-hardness.

- We propose:
  (i) a novel precision assignment method as a heuristic solution to the tradeoff problem;
  (ii) a novel technique that can handle too many overflows arising in training.

- We demonstrate that our techniques outperform existing precision assignments.

Summary of Contributions [Submitted]

- We formally introduce the memory-accuracy tradeoff problem and prove its NP-hardness.
- We propose:
  (i) a novel precision assignment method as a heuristic solution to the tradeoff problem;
  (ii) a novel technique that can handle too many overflows arising in training.
- We demonstrate that our techniques outperform existing precision assignments.

$$\mathrm{NN} \longrightarrow \boxed{\min_{w} \mathcal{L}(\mathrm{NN}_w)}$$

memory & time ⇓

Agenda

Programs that implement `math.h`.	Correctness: [PLDI'16], [POPL'18].
Probabilistic / differentiable programming.	Correctness: [NeurIPS'18/20], [POPL'20/23].
Programs that train ML models.	Acceleration: [Submitted].
Programs that compute derivatives.	Correctness: [ICML'23].

Autodiff

- Autodiff (AD): a class of algorithms that compute

$$\mathcal{D}P(x) \in \mathbb{R}^{m \times n} \quad \text{(when it exists)}$$

  for a given program $P : \mathbb{R}^n \to \mathbb{R}^m$ and input $x \in \mathbb{R}^n$, by applying the chain rule.

- Backpropagation algorithm: an instance of AD, widely used in machine learning.

Correctness of AD

matmul, sequential composition, …

- If P consists of differentiable functions and language constructs, then

$$\exists\ \mathcal{D}P(x) \quad \wedge \quad \mathcal{D}P(x) = \mathcal{D}^{\mathrm{AD}}P(x) \qquad \text{for all } x \in \mathbb{R}^n.$$

Correctness of AD

- If P consists of differentiable functions and language constructs, then

$$\exists\, \mathcal{D}P(x) \quad \wedge \quad \mathcal{D}P(x) = \mathcal{D}^{\mathrm{AD}}P(x) \qquad \text{for all } x \in \mathbb{R}^n.$$

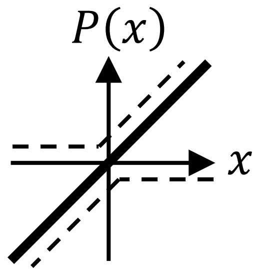
ReLU, if-else statement, …

- If P uses non-differentiable functions or language constructs, then

$$\nexists\, \mathcal{D}P(x) \quad \vee \quad \mathcal{D}P(x) \neq \mathcal{D}^{\mathrm{AD}}P(x) \qquad \text{for some } x \in \mathbb{R}^n.$$

E.g., for $P(x) = \mathrm{ReLU}(x) - \mathrm{ReLU}(-x)$,

$\mathcal{D}P(0) = 1$ but $\mathcal{D}^{\mathrm{AD}}P(0) = 0$.



Correctness of AD

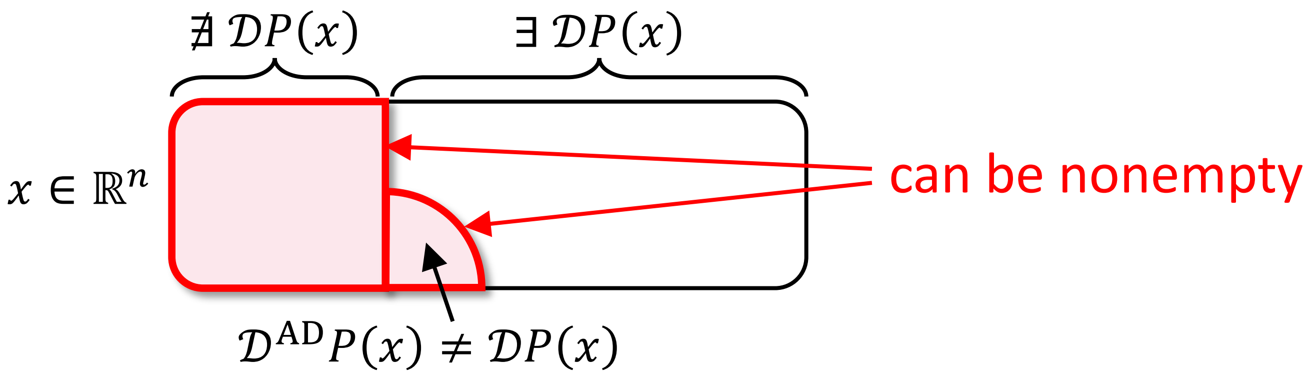
- If P consists of differentiable functions and language constructs, then

$$\exists\, \mathcal{D}P(x) \quad \wedge \quad \mathcal{D}P(x) = \mathcal{D}^{\mathrm{AD}}P(x) \qquad \text{for all } x \in \mathbb{R}^n.$$

- If P uses non-differentiable functions or language constructs, then

$$\nexists\, \mathcal{D}P(x) \quad \vee \quad \mathcal{D}P(x) \neq \mathcal{D}^{\mathrm{AD}}P(x) \qquad \text{for some } x \in \mathbb{R}^n.$$

That is,



Correctness of AD

- If P consists of differentiable functions and language constructs, then

$$\exists\, \mathcal{D}P(x) \quad \wedge \quad \mathcal{D}P(x) = \mathcal{D}^{\mathrm{AD}}P(x) \qquad \text{for all } x \in \mathbb{R}^n.$$

My previous result [NeurIPS'20]

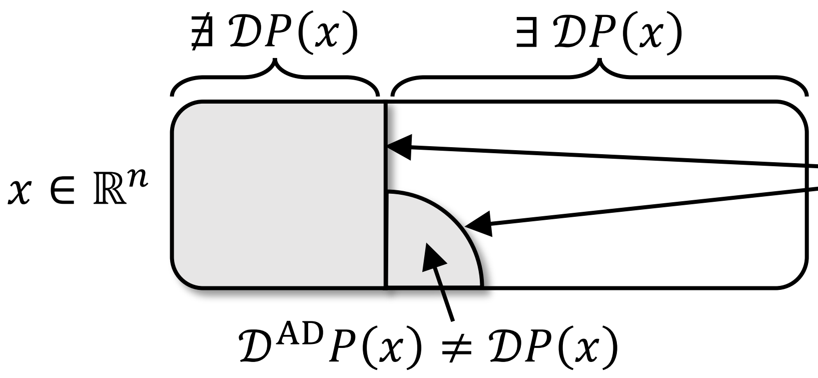
"piecewise analytic"

include ReLU, if-else statement, …

- If P uses ~~non-differentiable~~ functions or language constructs, then

$$\nexists\, \mathcal{D}P(x) \quad \vee \quad \mathcal{D}P(x) \neq \mathcal{D}^{\mathrm{AD}}P(x) \qquad \text{for some } x \in \mathbb{R}^n.$$

That is,



has measure zero (i.e., negligible)

~~can be nonempty~~

Limitations

- If P

My p

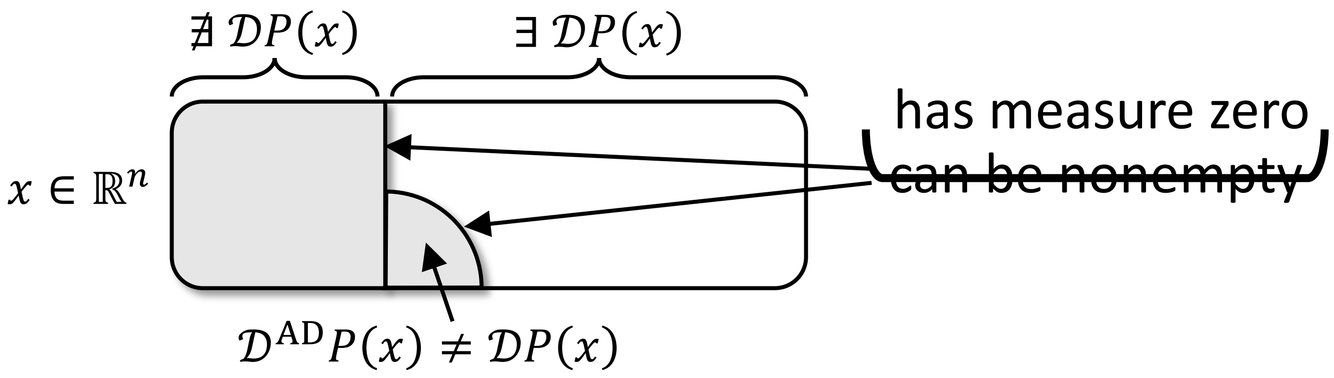
(1) In practice, inputs to programs are not reals, but often floats.

(2) The set of all floats $\mathbb{F}$ is finite, so has measure zero in $\mathbb{R}$.

- If P uses ~~non-differentiable~~ "piecewise analytic" functions or language constructs, then (e.g., ReLU, if-else statement)

$$\nexists\, \mathcal{D}P(x) \quad \vee \quad \mathcal{D}P(x) \neq \mathcal{D}^{\mathrm{AD}}P(x) \qquad \text{for } \cancel{\text{some}}\ \text{“negligible”}\ x \in \mathbb{R}^n.$$

That is,



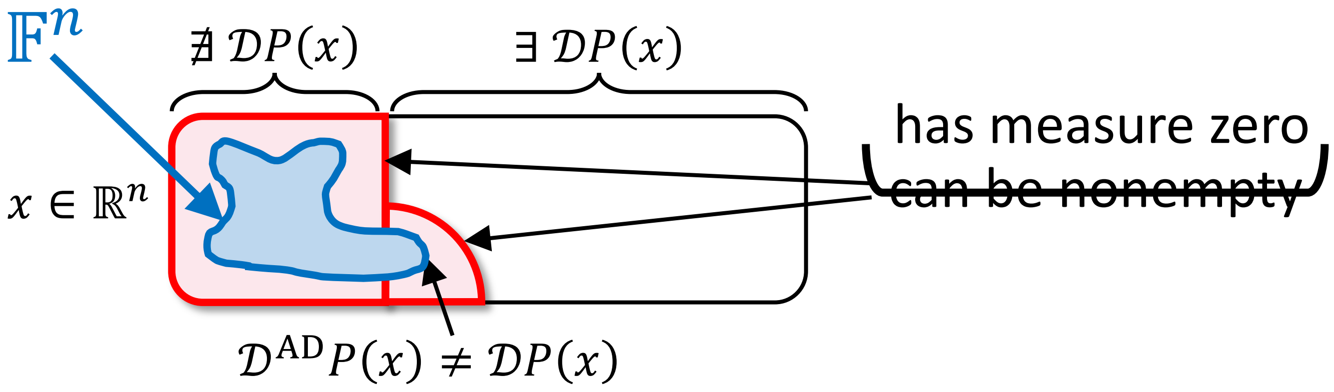
Limitations

(1) In practice, inputs to programs are not reals, but often floats.

(2) The set of all floats $\mathbb{F}$ is finite, so has measure zero in $\mathbb{R}$.

Hence, AD can be incorrect for all $x \in \mathbb{F}^n$, and this is indeed possible.

That is,



Problem

- Study the correctness of AD when inputs are floats (not reals).
- We focus on programs $P : \mathbb{R}^n \to \mathbb{R}^m$ that represent neural networks:

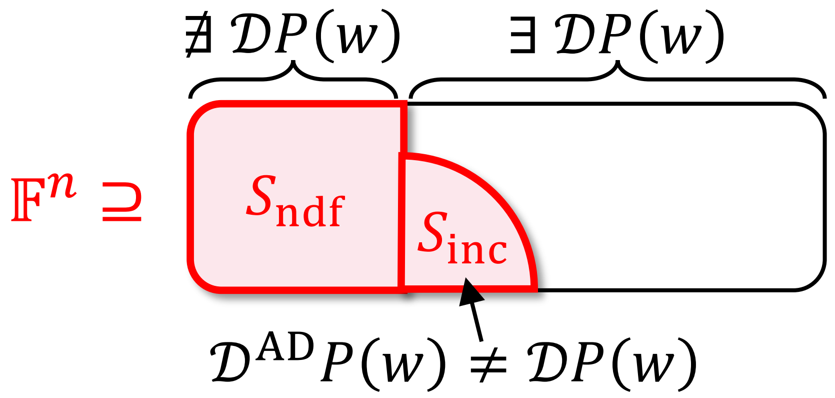
$$w \quad \longmapsto \quad P(w).$$

Problem

- Study the correctness of AD when inputs are floats (not reals).
- We focus on programs $P : \mathbb{R}^n \to \mathbb{R}^m$ that represent neural networks:

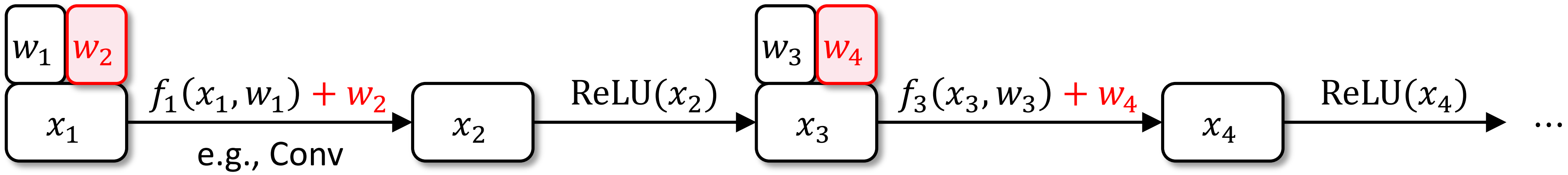
$$w \quad \longmapsto \quad P(w).$$

- Goal: Bound the size of the incorrect set (S_{inc}) and non-differentiable set (S_{ndf}) of P.



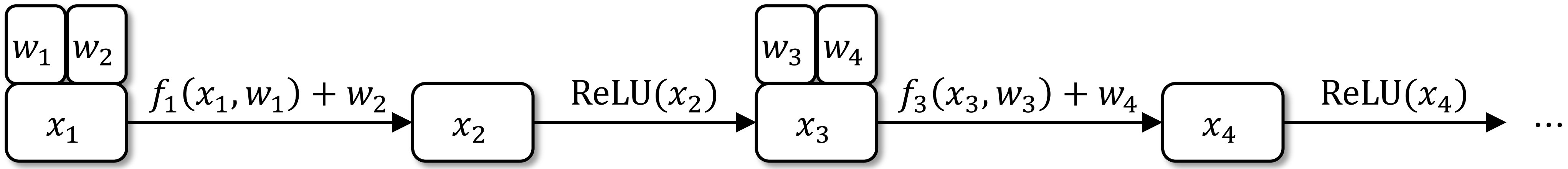
Our Results

- Consider a neural network P with bias parameters:



Our Results

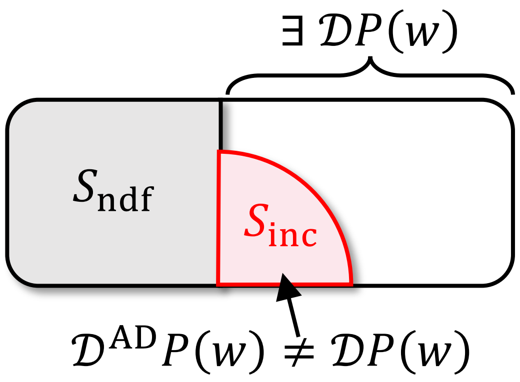
- Consider a neural network P with bias parameters:



- <u>Theorem 1</u> The incorrect set is always empty:

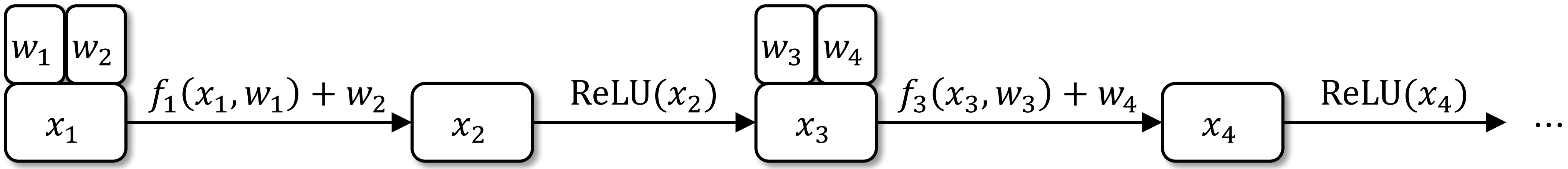
$$|S_{\mathrm{inc}}| = 0.$$

somewhat surprising, given that
there were no such type of results before



Our Results

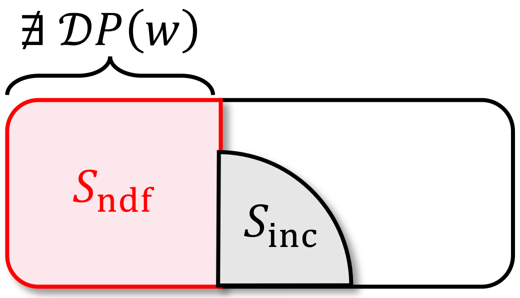
- Consider a neural network P with bias parameters:



- <u>Theorem 2</u> The density of the non-differentiable set is upper-bounded by

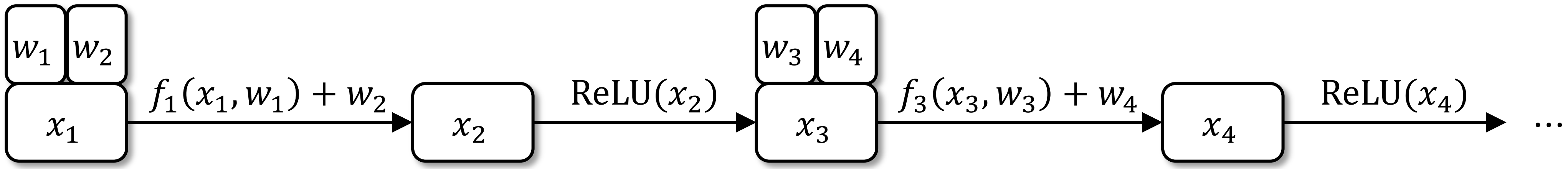
$$\frac{|S_{\mathrm{ndf}}|}{|\mathbb{F}^n|} \leq \frac{\#\ \mathrm{ReLUs\ in}\ P}{|\mathbb{F}|}.$$

2^{32} for 32-bit floats



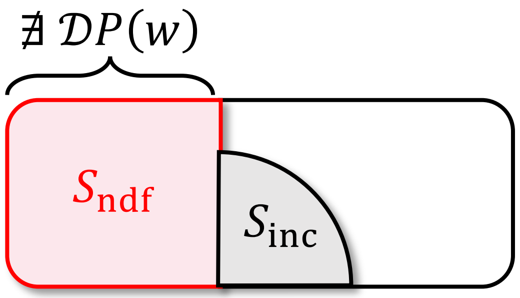
Our Results

- Consider a neural network P with bias parameters:



- Theorem 2 The density of the non-differentiable set is upper-bounded by

$$\frac{|S_{\mathrm{ndf}}|}{|\mathbb{F}^n|} \leq \frac{\#\ \mathrm{ReLUs\ in}\ P}{|\mathbb{F}|}.$$

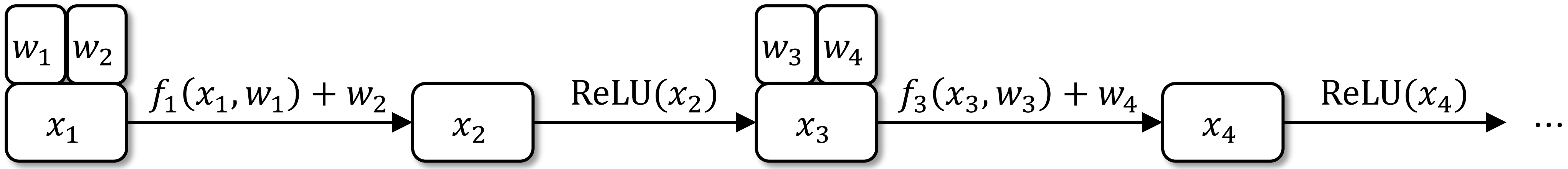


- Theorem 3 For many P, the above density is lower-bounded by

$$\frac{|S_{\mathrm{ndf}}|}{|\mathbb{F}^n|} \geq \frac{1}{2} \cdot \frac{\#\ \mathrm{ReLUs\ in}\ P}{|\mathbb{F}|}.$$

Our Results

- Consider a neural network P with bias parameters:

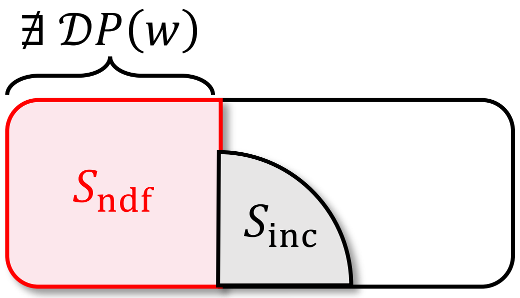


- Theorem 4 Over the non-differentiable set, AD computes a generalized derivative:

$$\mathcal{D}^{\mathrm{AD}} P(w) \in \partial P(w) \qquad \text{for all } w \in S_{\mathrm{ndf}}.$$

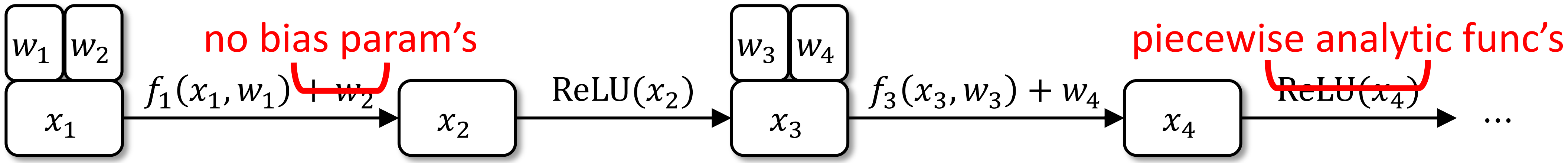
Clarke subdifferential of P

$$\triangleq \operatorname{conv}\left\{ \lim_{t\to\infty} \mathcal{D}P(w_t) : w_t \to w \right\}$$



Our Results

- Consider a neural network P with bias parameters:



- Theorem 3 On the non-differentiable set, AD computes a generalized derivative:

Extend previous results to more general neural networks.

Main point: Bounds become larger without bias parameters.

Summary of Contributions [ICML'23]

- We theoretically study the correctness of AD for neural networks when param's are floats.
- We prove tight bounds on the density of the incorrect and non-differentiable sets.
  We also prove what AD computes over these sets.
- Our results imply that AD for neural networks is correct on most floating-point param's,
  and it is correct more often with bias parameters.

Summary of Contributions [ICML'23]

- We theoretically study the correctness of AD for neural networks when param's are floats.
- We prove tight bounds on the density of the incorrect and non-differentiable sets.
  We also prove what AD computes over these sets.
- Our results imply that AD for neural networks is correct on most floating-point param's,
  and it is correct more often with bias parameters.

$$P \longrightarrow \boxed{\cdots \ \mathcal{D}^{\mathrm{AD}} P \ \cdots}$$

Agenda

Programs that implement `math.h`.	Correctness: [PLDI’16], [POPL’18].
Probabilistic / differentiable programming.	Correctness: [NeurIPS’18/20], [POPL’20/23].
Programs that train ML models.	Acceleration: [Submitted].
Programs that compute derivatives.	Correctness: [ICML’23].

$\Longrightarrow$ Have widened our understanding of floating point in real-world systems.

Questions?