∇*Fuzz*

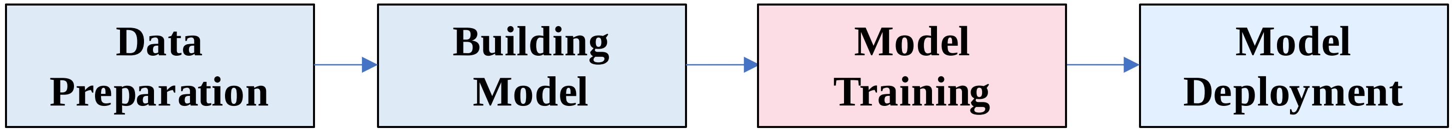
Fuzzing *A*utomatic *D*ifferentiation in Deep-Learning Libraries

Chenyuan Yang, Yinlin Deng, Jiayi Yao
Yuxing Tu, Hanchi Li, Lingming Zhang

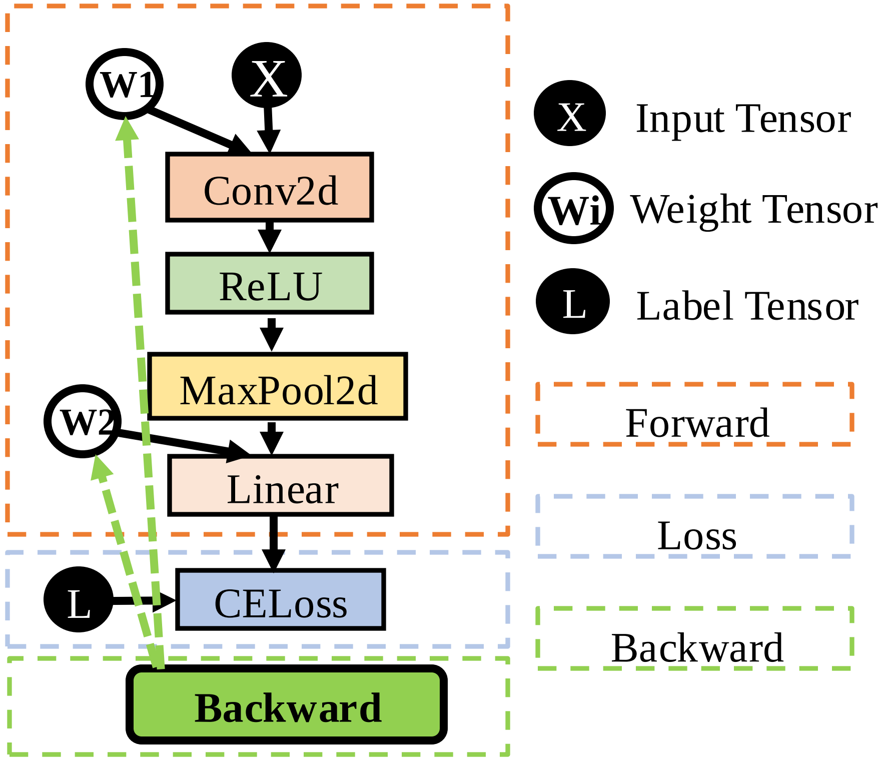
ICSE2023

Deep Learning (DL) Libraries

- DL pipeline



- DL libraries
  - Provide DL APIs for building models
  - Include an **Automatic Differentiation (AD) engine** for training the models



Automatic differentiation (AD) engine is a **crucial component** of any DL system.

Testing DL Libraries

- Model level fuzzers

input →  → output

- API level fuzzers

input → Conv2D → output

Prior work mainly focuses on inference phase
Testing the correctness of AD is still understudied

Backward Pass

- Compare the gradient given by multiple libraries
- Only covers reverse-mode AD
- Only covers 79 DL APIs with manual annotation
- Failed to detect any confirmed AD bug

[1]Gu *et al*. “Muffin: Testing deep learning libraries via neural architecture fuzzing”.

Bugs in AD engine

- Training a model is a resource-consuming process
- Imagine a bug in the middle…

AD bugs may cause DL models to crash, fail to converge, and/or perform poorly in practical deployment, which is **fatal** for safety-critical applications.

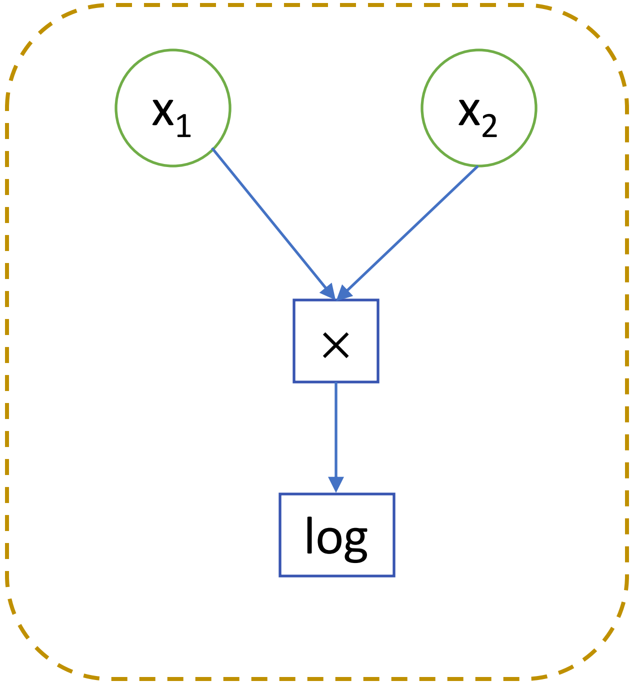
KLDivLoss is a very popular API, used in variational autoencoder (VAE), generative adversarial networks (GANs), recurrent neural networks (RNNs)

This bug[1] is found by us in PyTorch and labelled as **high priority**

[1] https://github.com/pytorch/pytorch/issues/78867

Differentiation

$$f(x_1, x_2) = \log(x_1 \cdot x_2)$$



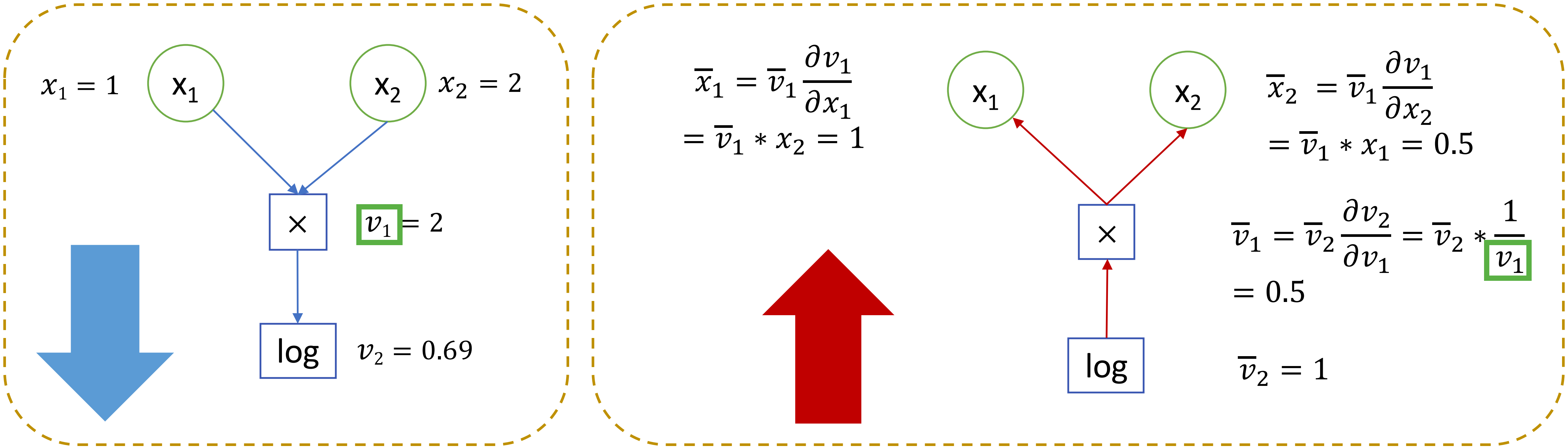
$$\frac{\partial f}{\partial x_1} = ?$$

- How to compute the partial gradient?
  - Automatic Differentiation (AD)
    - Reverse AD
    - Forward AD
  - Numerical Differentiation (ND)
    - ND

$$\frac{\partial f(\boldsymbol{x})}{\partial x_i} \approx \frac{f(\boldsymbol{x} + \delta \boldsymbol{e}_i) - f(\boldsymbol{x} - \delta \boldsymbol{e}_i)}{2\delta}$$

Differential Testing

Reverse Mode AD



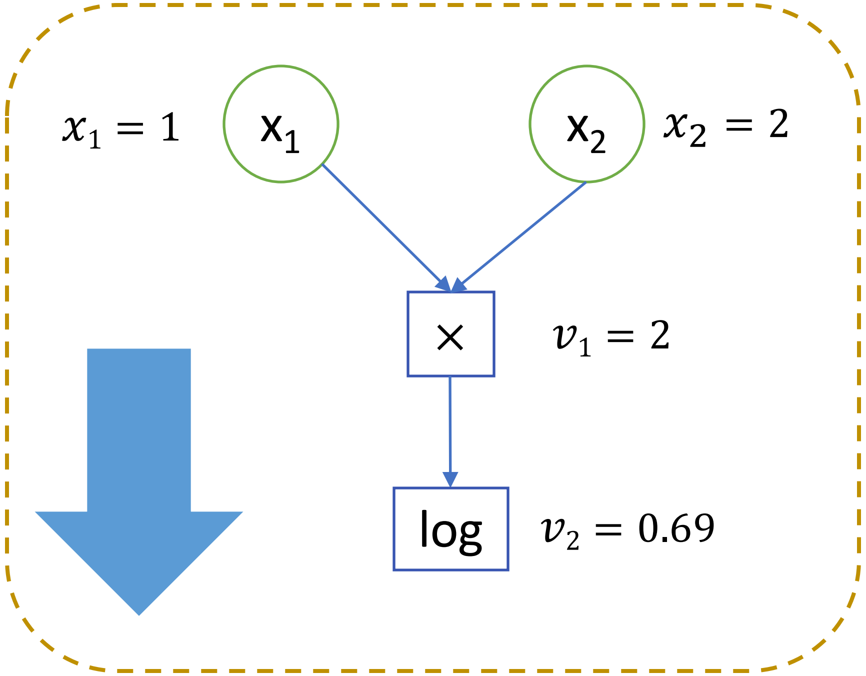
Forward Phase

Backward Phase

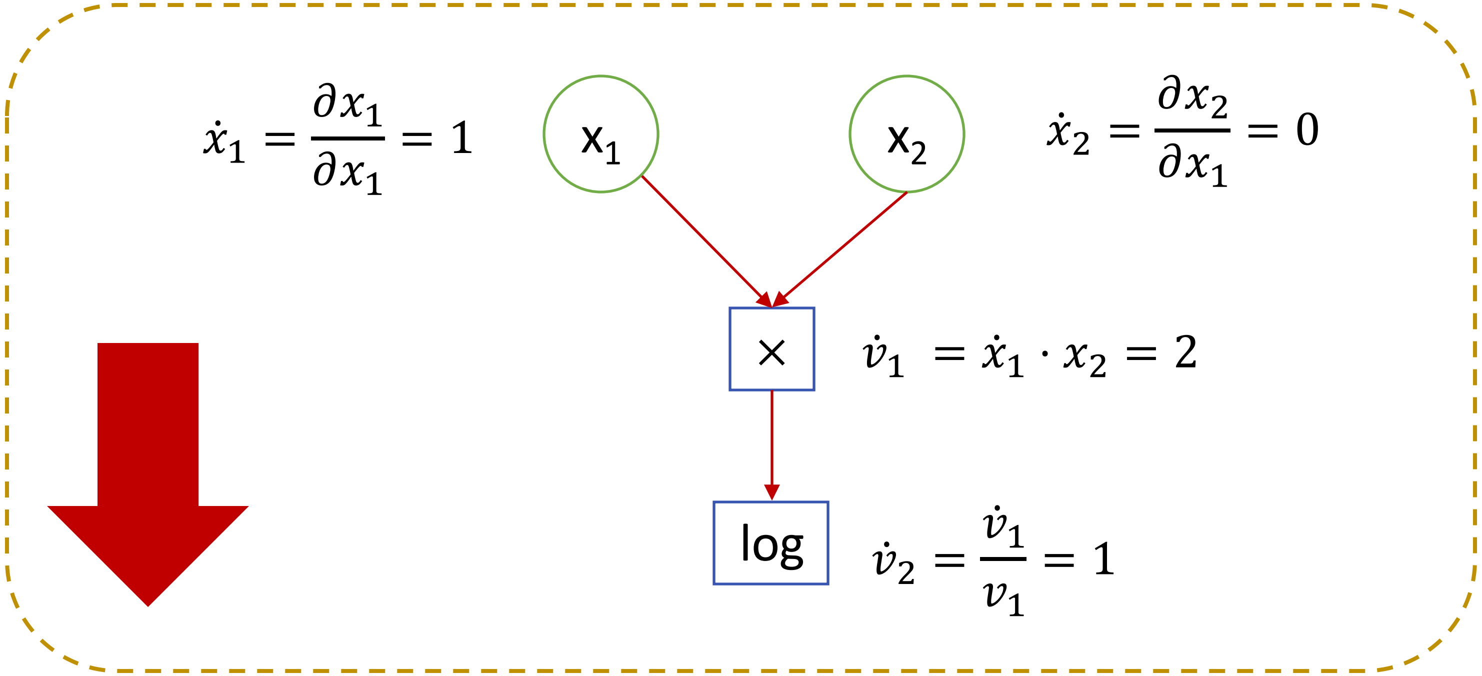
The **most common** AD mode in DL libraries

Efficient for high-dim input and low-dim output

Forward Mode AD



Forward Primal Phase



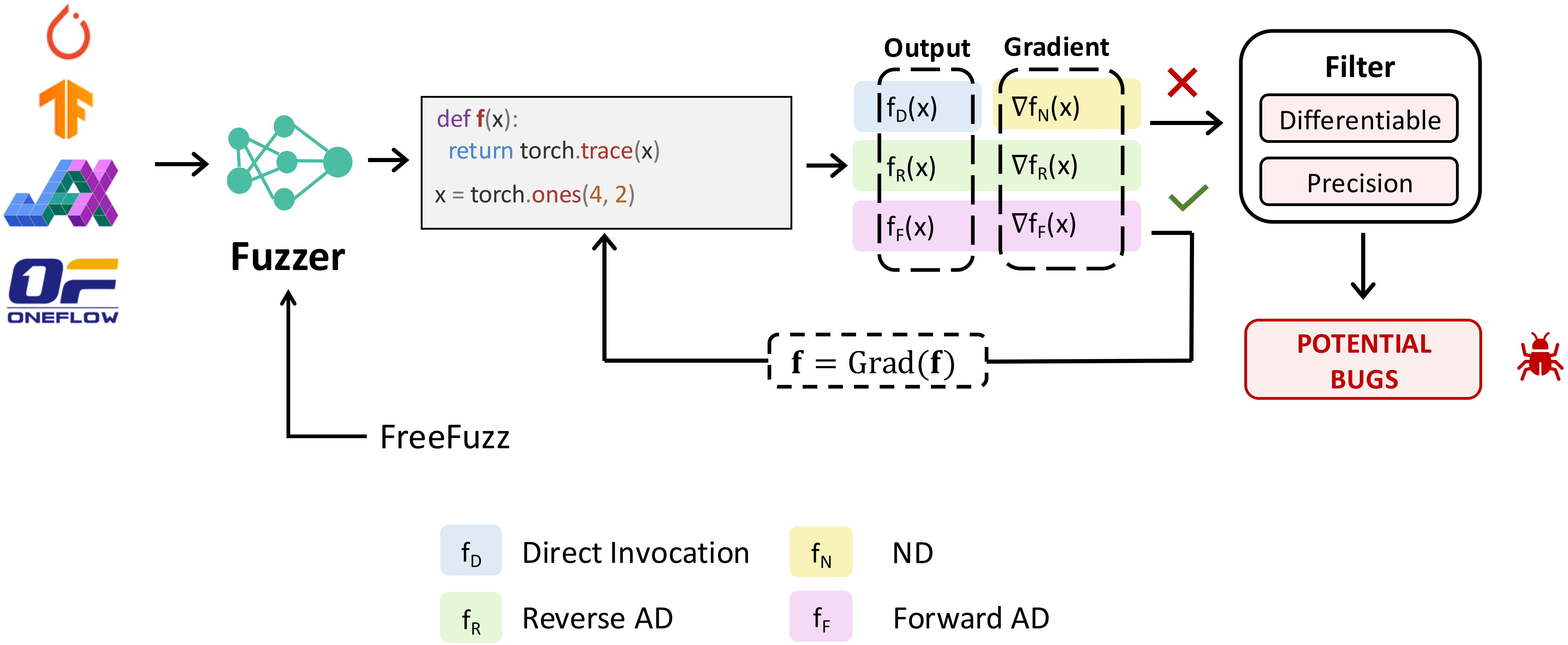
Forward Tangent Phase

Emerging: supported in TensorFlow and PyTorch (beta);
the **basis** of JAX's AD engine [1]
Efficient for high-dim output and low-dim input

[1] Radul *et al*. "You Only Linearize Once: Tangents Transpose to Gradients". POPL 2023.

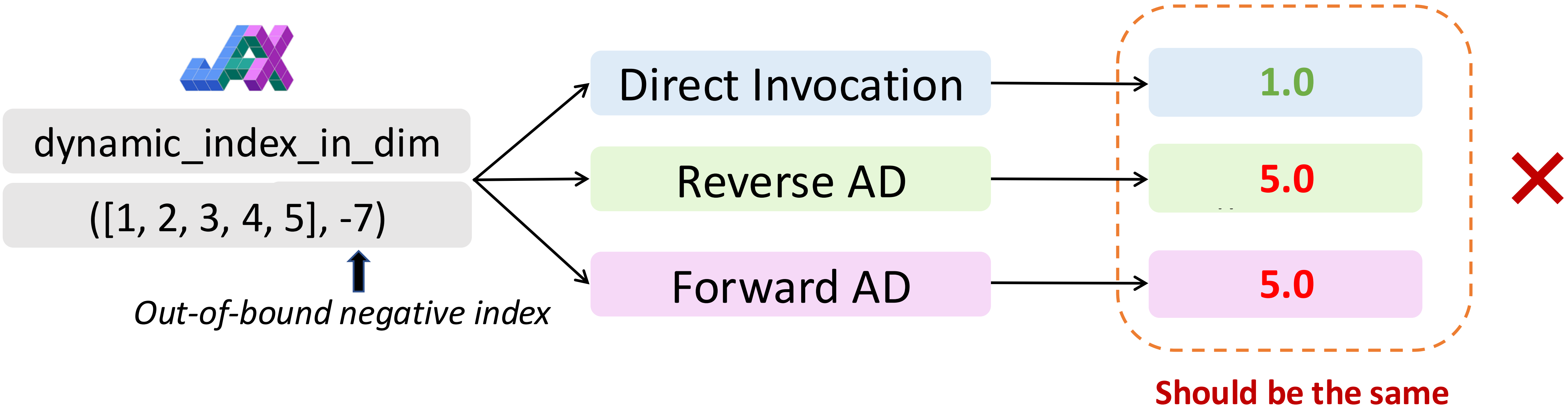
Framework of ∇Fuzz

- The first approach specifically targeting the AD engine in DL libraries



Oracle: Output Check

- Compare the output of invocation in different execution scenarios
  - When calculating the gradient, additional operations are incurred
  - Despite different implementation, the output should be the same

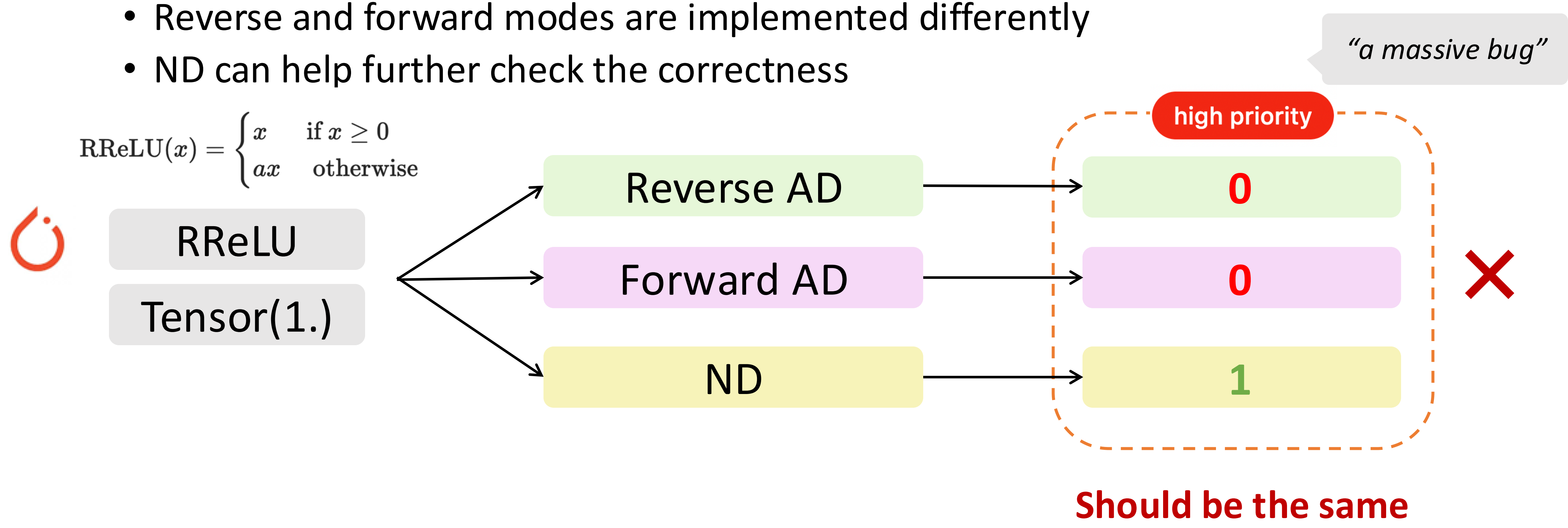


A bug detected by ∇Fuzz and fixed in JAX

Oracle: Gradient Check

- Compare the gradient in different execution scenarios
  - Reverse and forward modes are implemented differently
  - ND can help further check the correctness

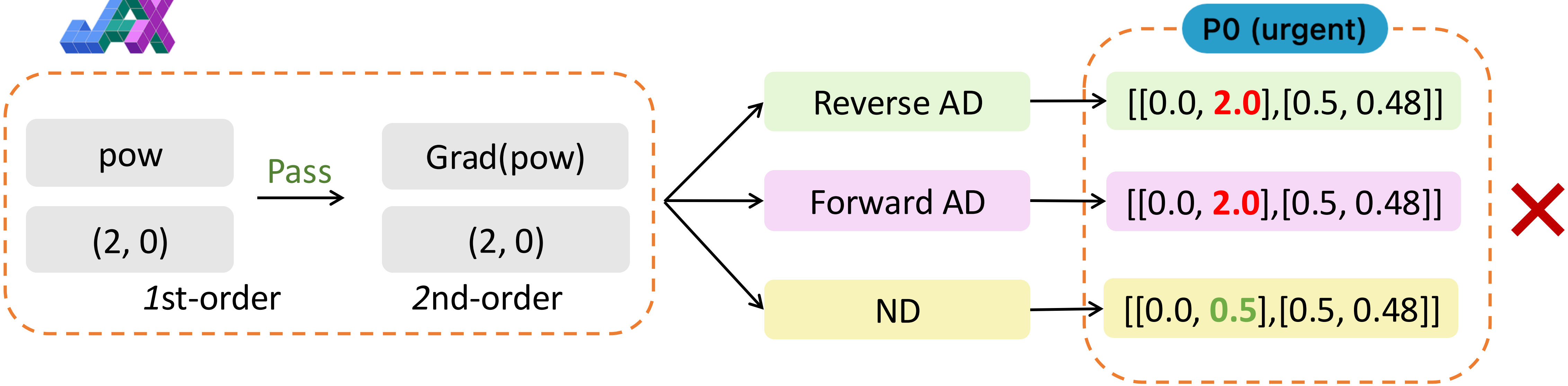
$$\mathrm{RReLU}(x) = \begin{cases} x & \text{if } x \geq 0 \\ ax & \text{otherwise} \end{cases}$$



A bug detected by ∇Fuzz and fixed in PyTorch

High-order Gradient

- ∇Fuzz oracle can run on the *gradient function* of current function
  - It can test any order of gradient function



A bug detected by ∇Fuzz and fixed immediately in JAX **Should be the same**

- **First- and second-order** gradient computation are the most frequently used

Evaluation: Bug Detection

- ∇Fuzz detects 173 bugs in total
  - 144 confirmed
    - 117 previously unknown
    - 107 are AD-related

Contributed **58.3% (7/12)** of all *high-priority AD bugs* for PyTorch and JAX during a two-month period.

None of the 107 AD-related bugs can be detected by existing work.

Library	Total	Confirmed (Fixed)		Won't Fix
		Unknown	Known	
PyTorch	80	62 (10)	15 (9)	3
TensorFlow	29	18 (0)	5 (2)	2
JAX	34	20 (5)	3 (2)	1
OneFlow	30	17 (6)	4 (4)	0
Total	**173**	**117 (21)**	**27(17)**	**6**

Fuzzing Automatic Differentiation in Deep-Learning Libraries

- **∇Fuzz: first approach specifically targeting the *AD engine* in DL libraries,** which is a crucial component of any DL system
  - Leverage different execution scenarios as test oracles to differentially test first- and high-order gradients
  - The core ∇Fuzz idea is general and can be used as oracle for future fuzzers at different levels (API or model levels)
- **Detected 173 bugs for PyTorch, TensorFlow, JAX, and OneFlow**
  - with 144 confirmed, 117 previously unknown, and 38 already fixed
  - Contributed 58.3% (7/12) of all high-priority AD bugs for PyTorch and JAX during a two-month period

Chenyuan Yang
cy54@illinois.edu

Yinlin Deng
yinlind2@illinois.edu

Tool: https://github.com/ise-uiuc/NablaFuzz

Back-up slides

Evaluation: Distribution of Confirmed Bugs

- Symptoms of confirmed bugs

∇Fuzz	Output	Gradient		
		Total	1st-order	2nd-order
PyTorch	31	46	44	2
TensorFlow	4	19	17	2
JAX	14	9	8	1
OneFlow	16	5	2	3
Total	**65**	**79**	**71**	**8**

More than half bugs are detected by inconsistent gradients.

Evaluation: Distribution of Confirmed Bugs

- Scenario distribution of confirmed bugs

Most of the bugs detected by ∇Fuzz are related to our main target AD

∇Fuzz	Direct	AD			ND
		All	Rev-Only	Fwd-Only	
PyTorch	11	64	33	9	2
TensorFlow	3	18	5	4	2
JAX	3	20	3	1	0
OneFlow	16	5	5	N/A	N/A
Total	**33**	**107**	**46**	**14**	**4**

Evaluation: FPR and Filter

	Output	Gradient				Total
		Diff+Precision	Diff	Precision	N/A	
Pytorch	19.3%	21.2%	25.5%	57.3%	61.9%	20.7%
Tensorflow	8.3%	21.1%	34.8%	46.4%	53.1%	16.1%
JAX	11.1%	21.0%	58.1%	68.6%	78.2%	17.3%
OneFlow	12.5%	25.0%	25.0%	64.0%	64.0%	20.0%
Total	**15.0%**	**21.3%**	**38.2%**	**60.6%**	**67.6%**	**19.3%**

With Filter

No Filter

Our filtering strategies reduce FPR from 67.6% to 21.3%.
Both filtering strategies are effective; differentiability is more helpful.

Example of Rejected Bug

- When x has the lowest precision floating datatype bfloat16
  - Inconsistent gradients by reverse and forward modes
  - It was rejected: *"This is a consequence of the intended design of bfloat16. It is a worthwhile tradeoff for speed in deep learning contexts..."*

$$\boldsymbol{sinc}(x) = \frac{\sin \pi x}{\pi x}$$

```
x = array(-0.125, dtype=bfloat16)
RevGrad(jax.numpy.sinc, x) # 0.34375
FwdGrad(jax.numpy.sinc, x) # 0.375
```

Evaluation: Coverage Comparison

	PyTorch			TensorFlow		
	C++ Cov	Python Cov	Time	C++ Cov	Python Cov	Time
FreeFuzz	70639 (21.1%)	14579 (13.9%)	3.1h	36279 (9.77%)	80220 (30.1%)	3.9h
∇Fuzz	86459 (25.8%)	15042 (14.3%)	25.7h	42284 (11.4%)	88783 (33.3%)	24.3h
∇Fuzz (seed only)	79808 (23.4%)	14854 (14.1%)	1.4h	37233 (10.0%)	84848 (31.9%)	2.9h

- Gradient computation is expensive (higher time cost).
- Gradient computation is important for system coverage:
  - ∇Fuzz (seed only) have higher code coverage even with less time compared to FreeFuzz.

Evaluation: Coverage Comparison

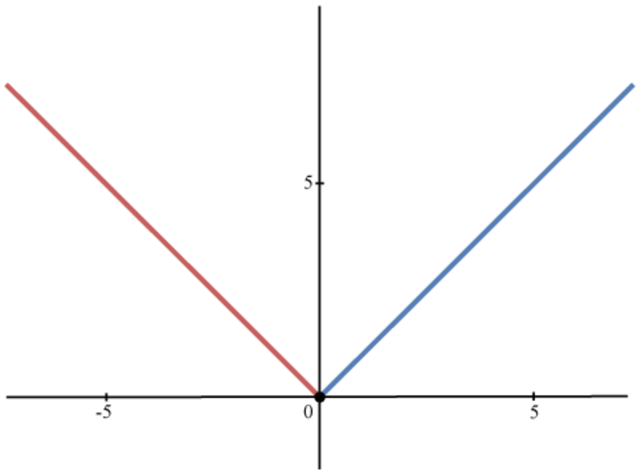
	C++ Coverage	Python Coverage	API Coverage	Time	Rev AD	Fwd AD	ND
∇Fuzz	41625 (11.21%)	88524 (33.24%)	1902	6.1h	✓	✓	✓
Muffin	36884 (9.94%)	78754 (29.57%)	79	6.8h	✓		

∇Fuzz substantially outperforms Muffin in both code and API coverage, with slightly less execution time.
∇Fuzz can thoroughly and automatically test the AD engines.

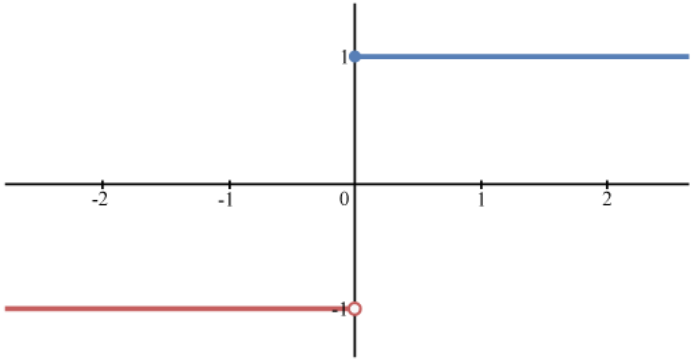
*For fair comparison, we run ∇Fuzz by setting the number of mutants for each API to 150

Filter: Differentiability

- Non-differentiable
  - The gradient at the non-differentiable point is *undefined*
- Differentiability
  - **f** is continuous at **x**, and
  - All partial derivatives of **f** exist in the neighborhood of **x** and are continuous at **x**
- Sample neighbors of x and compare the output and gradient
  - Leverage ND



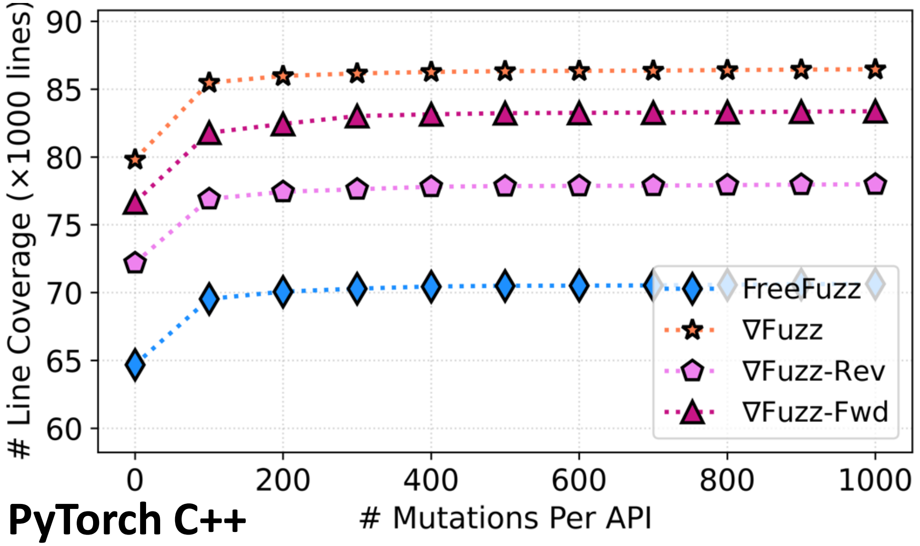
$$f(x) = |x| = \begin{cases} x & x \geq 0 \\ -x & x < 0 \end{cases}$$



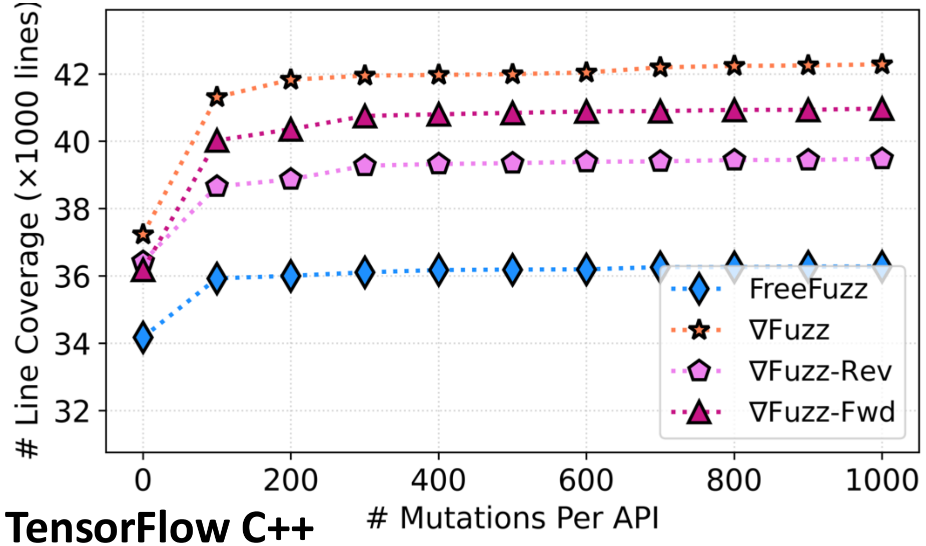
$$f'(x) = \begin{cases} 1 & x \geq 0 \\ -1 & x < 0 \end{cases}$$

Evaluation: Coverage

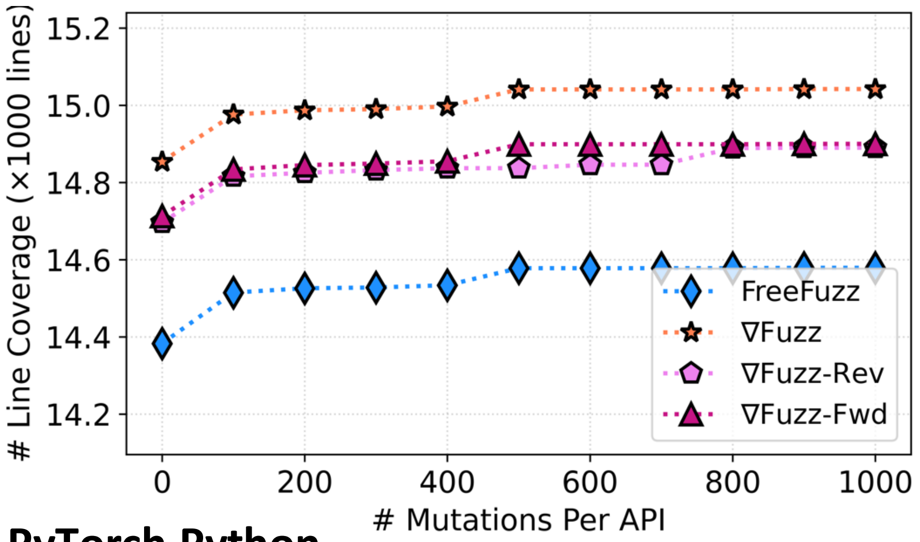
∇Fuzz-Rev (**disabling** reverse mode AD)
∇Fuzz-Fwd (**disabling** forward mode AD)



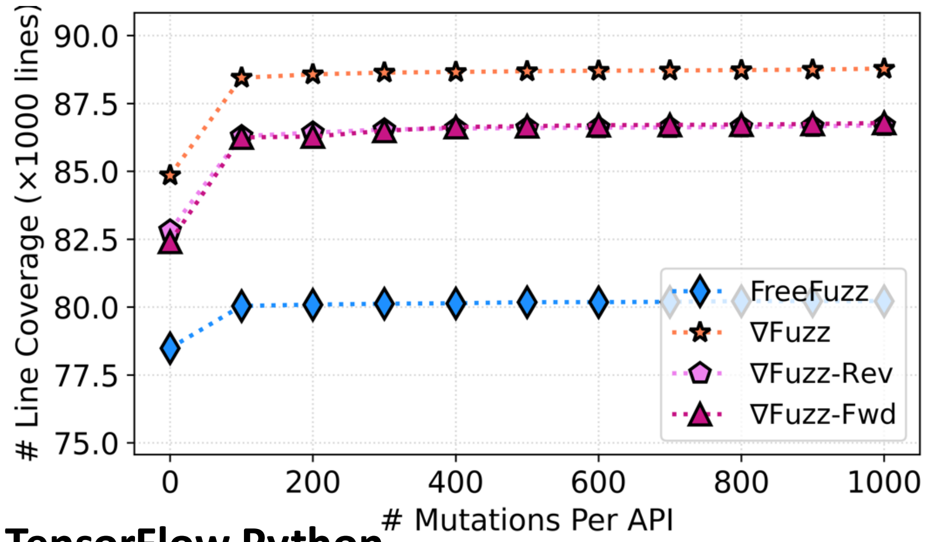
PyTorch C++



TensorFlow C++



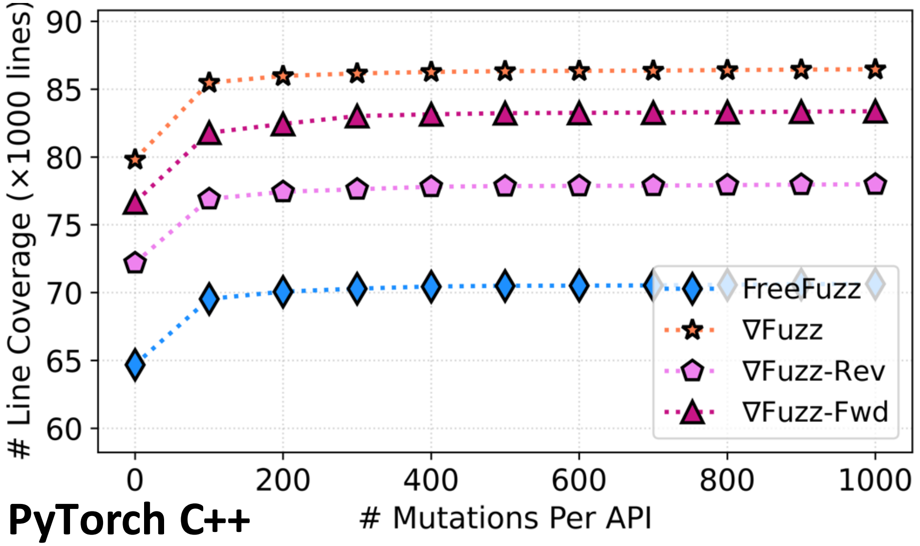
PyTorch Python



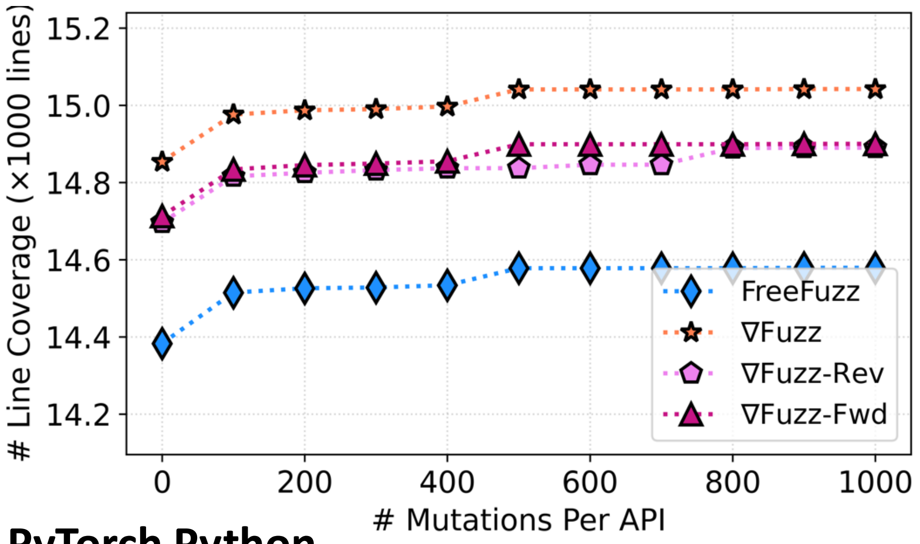
TensorFlow Python

Evaluation: Coverage

∇Fuzz-Rev (**disabling** reverse mode AD)
∇Fuzz-Fwd (**disabling** forward mode AD)



PyTorch C++



PyTorch Python

- For C++ coverage, ∇Fuzz outperforms FreeFuzz significantly on both PyTorch and TensorFlow , with an improvement of **22.4%/16.6%**.
- ∇Fuzz has larger improvement on **C++** coverage than Python
  - *"Autograd is a hotspot for PyTorch performance, so most of the heavy lifting is implemented in C++"*
- Disabling **Rev** will hurt the performance most
  - *Reverse mode AD is the main technique and occupies a larger portion of the DL library implementation than the forward mode*