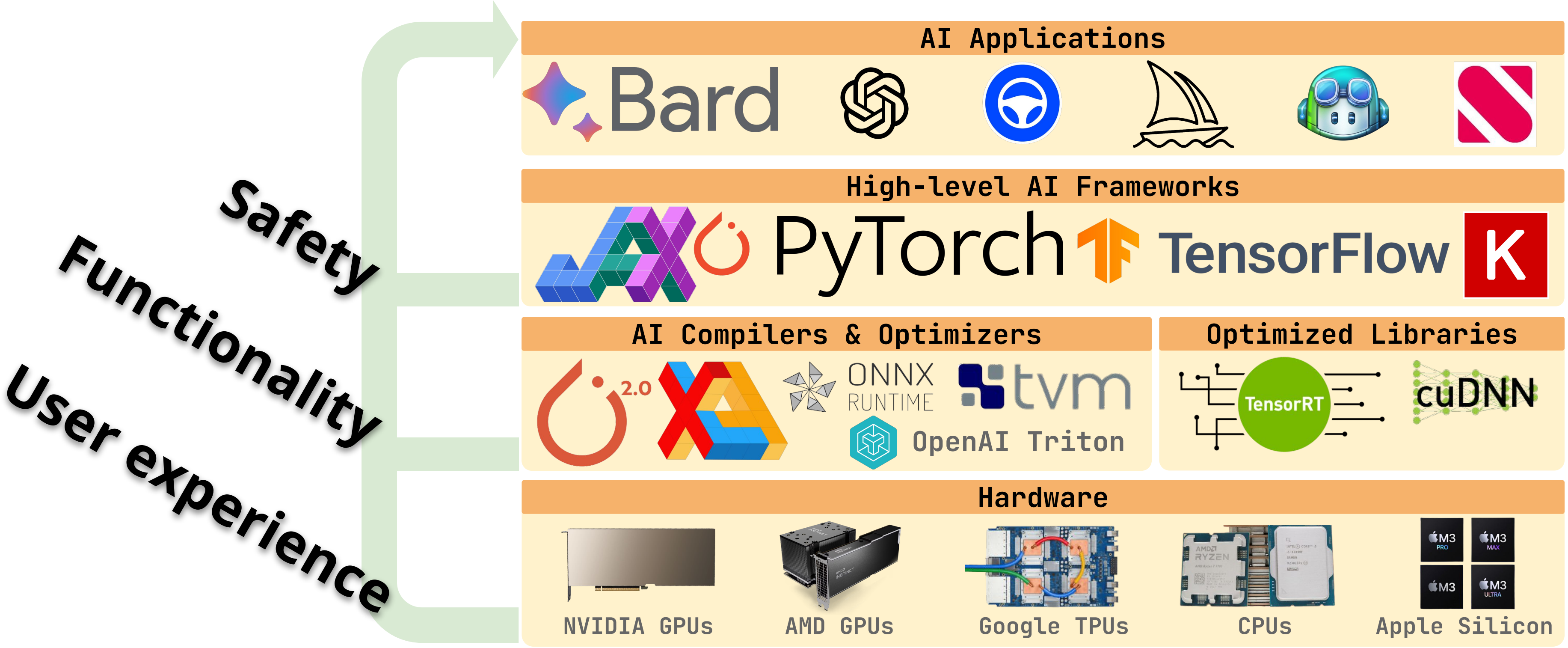
NeuRI: Diversifying DNN Generation via Inductive Rule Inference

Jiawei Liu, Jinjun Peng, Yuyao Wang, Lingming Zhang

ESEC/FSE 2023 @ San Francisco

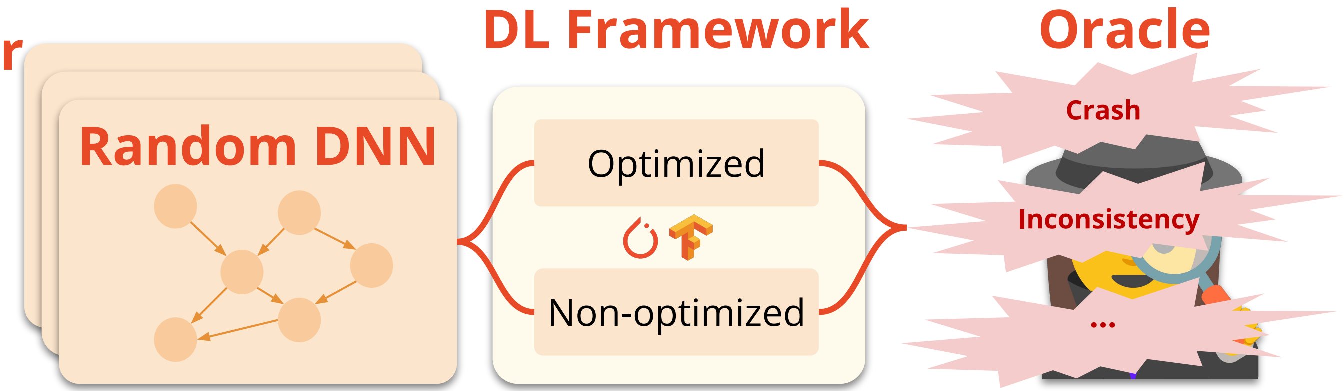
UNIVERSITY OF ILLINOIS URBANA-CHAMPAIGN
COLUMBIA UNIVERSITY IN THE CITY OF NEW YORK
南京大學 NANJING UNIVERSITY

DL System Correctness is Crucial



Generating Models as Tests

Test Generator
=
Model Generator



NeuRI [This work]
NNSmith [ASPLOS 23]
Muffin [ICSE 22]
...

How to Generate Valid Models?

Generating Valid Models

- **DNN model:** a directed graph of operators
- **Operator:** a function transforming tensors to tensors
- **Model validity** requires each operator to be
  - Legally constructed
  - Taking inputs of reasonable shapes/dimensions
  - Different operators have different constraints

Invalid Model

ksize larger than input sizes

```
x = … # shape=[1,3,32,32]
y = avg_pool(x, ksize=33)
```

Solver-aided Model Generation

A constraint solving approach by NNSmith [ASPLOS 23]

- **Define** composable constraints for each operator
- **Accumulate & solve** model-wise constraints

```
x= input() # [x0,x1,x2]
y= relu(x) # [y0,y1,y2]
z= pool(y, ksize,stride)
```

NNSmith

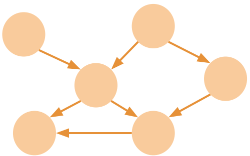
Collect Constraints

```
[x0,x1,x2] =x.shape >0
[y0,y1,y2] =y.shape =x.shape
(y1-ksize)//stride > 0
(y2-ksize)//stride > 0
...
```

Solve Constraints

Z3

```
{x0=1, x1=8, x2=8, ksize=3,...}
```

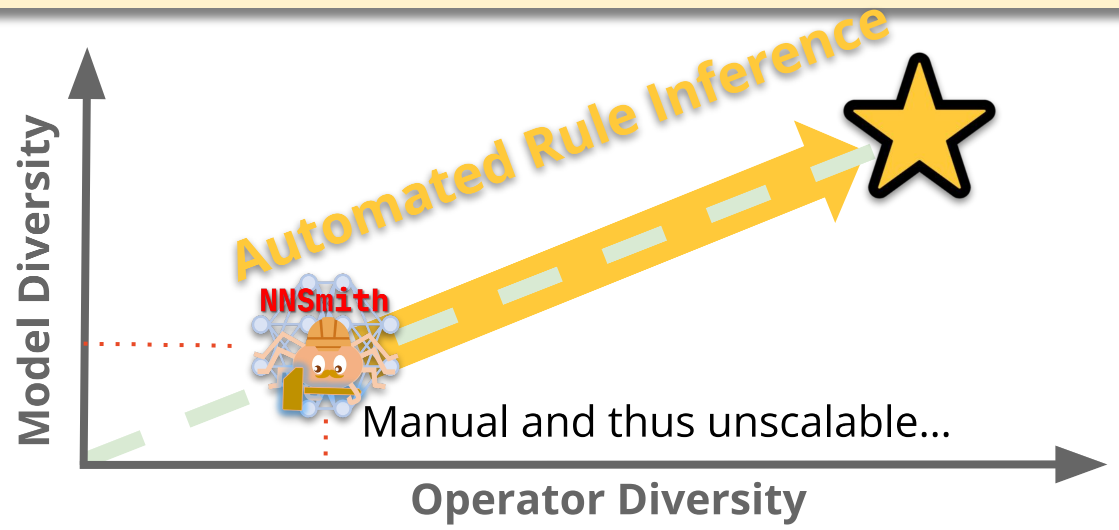


Concrete DNN

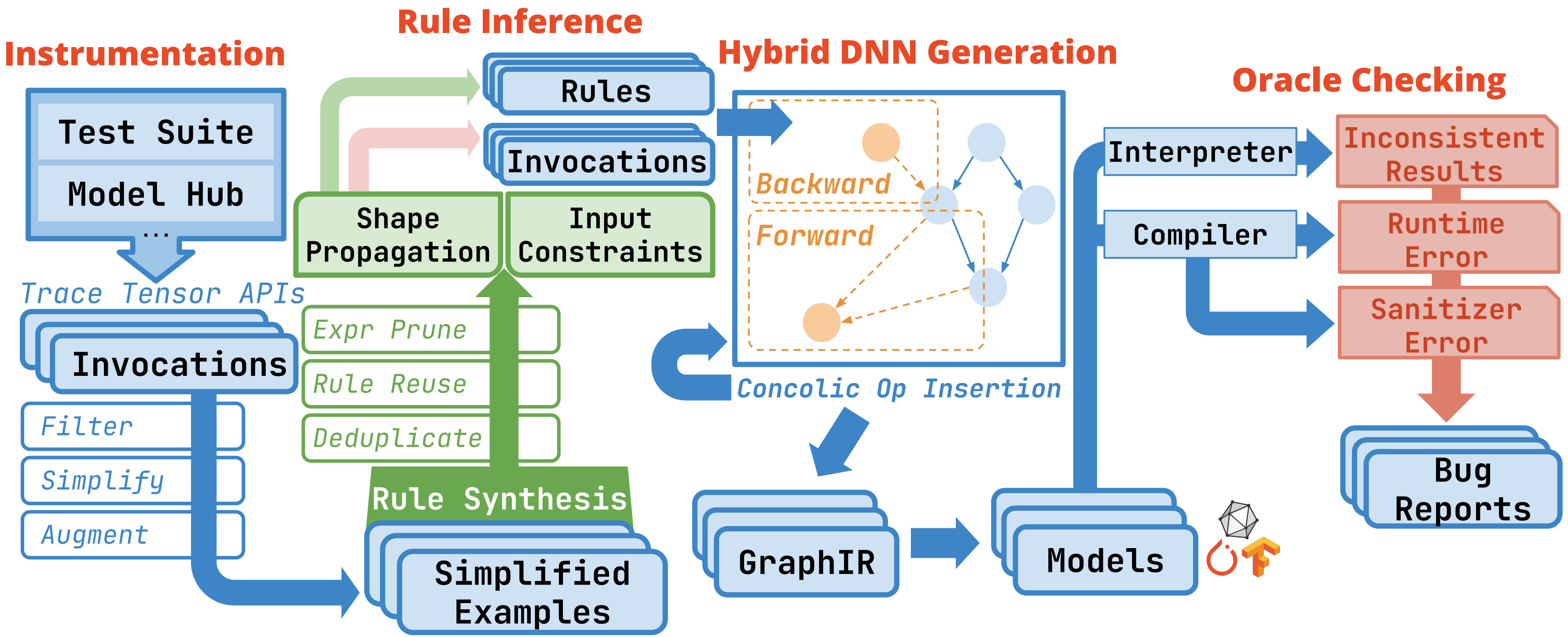
Diversifying Valid Models

- **Model diversity** is determined by **operator diversity**
- NNSmith manually supports **~60** operator rules

Can we *automatically* synthesize operator rules?

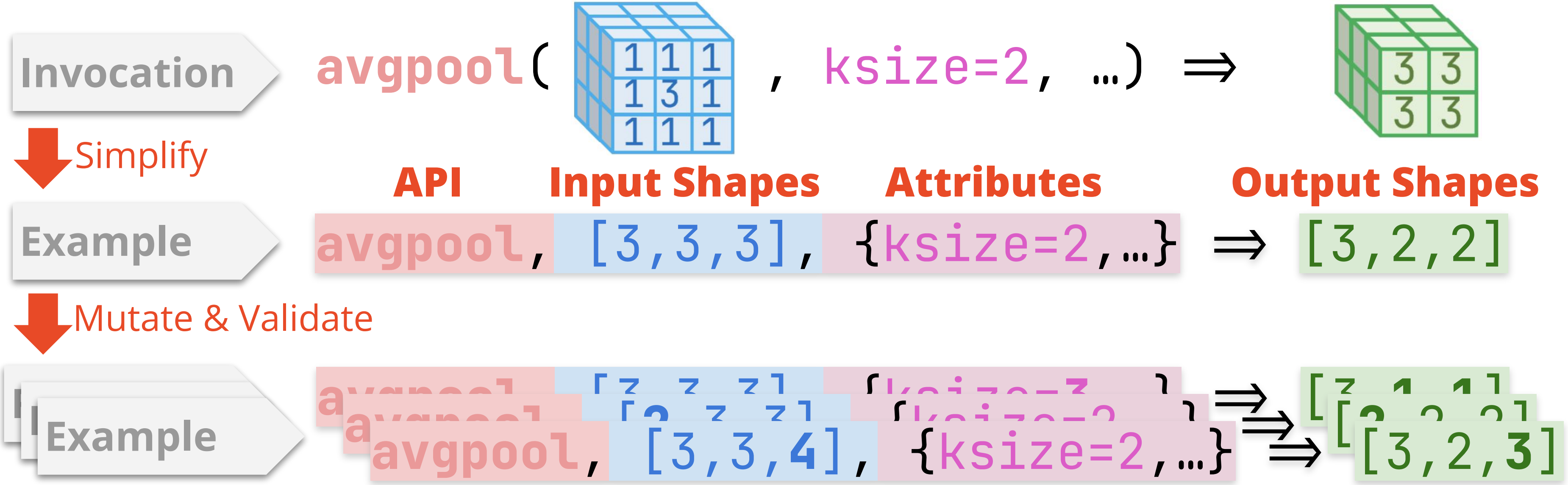


NeuRI: Neural Net Rule Inference



Instrumenting Concrete Operator Invocation

- Instrument operator invocations from regression tests
- Simplify the layout of invocations
- Create more records via mutation



Inferring Operator Rules from Records

Each type (e.g., operator) of records has **3 sets of symbols**

Input Shapes **I** Attributes **A** Output Shapes **O**

`avgpool, [3,3,4], {ksize=2,…}` ⇒ `[3,2,3]`

- **#1 Shape propagation:** $\{o = f(A \cup I);\ o \in O\}$
  - $\{O_0=I_0,\ O_1=(I_1-ksize)//stride+1,\ O_2=(I_2-ksize)//stride+1\}$
  - #constraints = #output dimensions (|**O**|)
- **#2 Input constraints:** $\{0 =/< f(A \cup I);\ …\}$
  - $\{ksize>0,\ stride>0,\ O_*>0,\ ...\}$
  - #constraints is variadic/unknown

Inductive Rule Inference

Let `f(A ∪ I)` be an expression under arithmetic grammar

```
⟨expr⟩    ::= ⟨op⟩ ⟨expr⟩⟨expr⟩ | ⟨item⟩
⟨item⟩    ::= ⟨symbol⟩ | ⟨literal⟩
⟨op⟩      ::= + | - | × | ÷ | min | max | mod
⟨symbol⟩  ::= Symbols from A ∪ I
⟨literal⟩ ::= Constant integers
```

Search-based Inductive Synthesis: Enumerate all terms of the grammar s.t. ∃ expr *satisfies* all collected examples

Optimization: Pruning the Search Space

We **prune** the search space of possible term skeletons by

- **Bounded search**: limit the AST depth & `⟨literal⟩`
- Prune **semantically equivalent** term skeletons
- **Rarity pruning:**
  - Skip **constant sub-term** `⟨op⟩ ⟨literal⟩⟨literal⟩`
  - One symbol only occur once in a term

- **Output** is a set of term skeletons pruned ahead of time
- At inference time, we substitute the holes in the skeleton → actual symbols for each type of records

More Optimizations

- **Rule reusing**
  - **Insight:** Operator rules can share similar patterns
  - Before rule synthesis, try if the records match any of the inferred rules
- **Post deduplication**
  - Inferred constraints are boilerplate: (i) not readable and (ii) inefficient when used in online solving
  - Example: `{a + b + 1 > 0, a + b > 0}` → `{a + b > 0}`

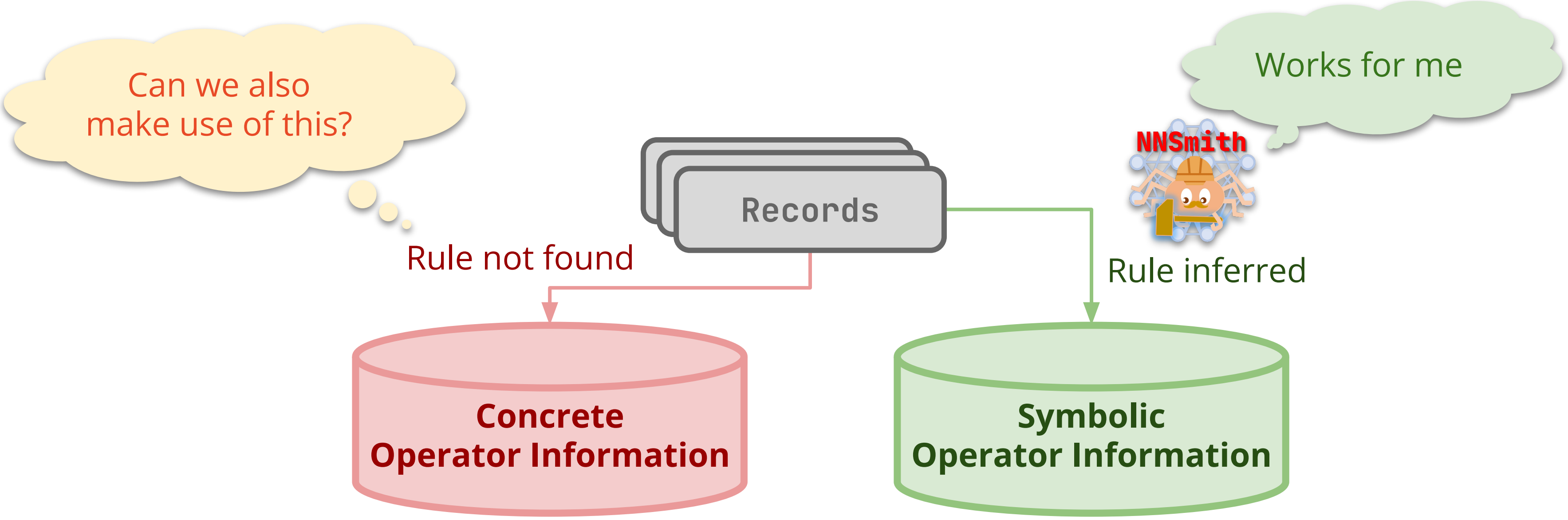
Given a set of *predicate* terms C, perform:

$$C = C - \{c\} \textit{ iff } \mathbf{conj}[C] \Leftrightarrow \mathbf{conj}[C - \{c\}]$$

until a fixed point

Model Generation

- Some rules are inferred and others are not
- NNSmith only works for symbolic operator (rule inferred)



Concolic Model Generation

Using both **concrete** + **symbolic** (concolic) information

- Constructing a graph ← Inserting an operator
- Inserting a **concrete** operator
  - Find invocations with exact shape match

```
x= input()      # [3,16,16]
y= pool(x,...) # [3,14,14]
```

Query invocation w/ input shape [3,14,14]

Concrete Operator Information

```
attn({},...)
```

```
x= input()      # [3,16,16]
y= pool(x,...) # [3,14,14]
z= attn(y,...) # inserted
```

Concolic Model Generation

Using both **concrete** + **symbolic** (concolic) information

- Constructing a graph ← Inserting an operator
- Inserting a **concrete** operator
  - Find invocations with exact shape match
- Inserting a **symbolic** operator
  - Solve the constraints immediately to make the graph *fully concrete*

Evaluation Setup

Systems under Test

PyTorch
- Torch Inductor
- Torch JIT

TensorFlow
- XLA
- TensorFlow Lite

NNSmith ASPLOS 23

Muffin ICSE 22

Variants of NeuRI

DeepREL FSE 22

Model-Level Fuzzer

Op-Level Fuzzer

Finding 100 Bugs in Four Months

- 🔥 **51** bugs fixed; **81** bugs fixed or confirmed
- 🔥 **9** bugs are marked as PyTorch high priority
- 🔥 **1** security vulnerability Moderate 6.3 / 10

Bug reports

> *"... the bugs you've reported are **high quality** ... don't look like specially fuzzed set that's impossible to see in **practice**. They did **reveal a few common themes** that are easy to encounter in **practice** ..."*
>
> -- PyTorch Developer (#93357)

@JiaweiLiu_ jiawei6@illinois.edu

Result Highlights

- **24% / 15%** coverage improvement over SOTA NNSmith
- **95% / 99%** generated (5-node) models are valid
- **~100ms** to generate and run a model on a single thread
- **4.6k rules** inferred by NeuRI in **1s** while Rosette…

Type	<1s	<10s	<100s	<1000s
NeuRI	4,660	4,700	4,716	4,758
Rosette	0	83	2,832	4,461

A lot more interesting results detailed our paper!

Summarizing NeuRI

- **Automatically discovering operator rules!**
  - Collecting input-output examples via *instrumentation* + *mutation*
  - Efficient *inductive program synthesis* with domain optimizations
  - *Concolic generation* to maximize both symbolic & concrete information
- Found **100** bugs including high-priority & security ones!
- Everything open-sourced!

Paper　　Code　　Bug reports