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

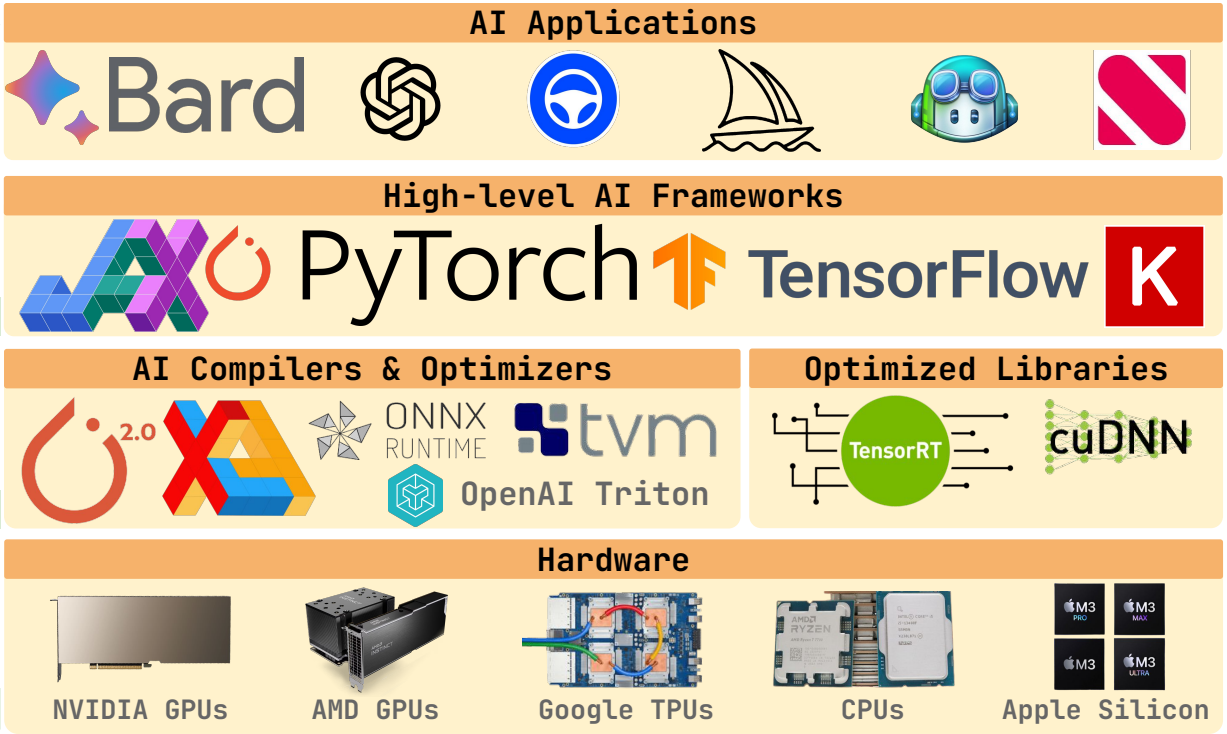
COLUMBIA UNIVERSITY IN THE CITY OF NEW YORK

南京大学 NANJING UNIVERSITY

@JiaweiLiu_ jiawei6@illinois.edu

DL System Correctness is Crucial

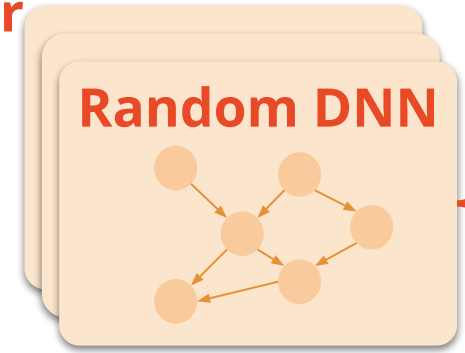
Safety
Privacy
Functionality
User experience



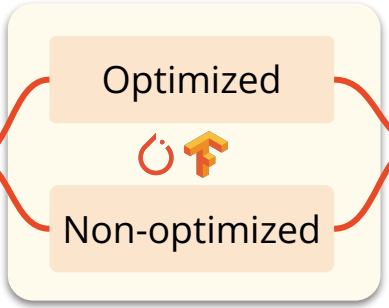
Generating Models as Tests

Test Generator
=
Model Generator

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



DL Framework



Oracle

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
  - Well-formedly constructed
  - Taking inputs of reasonable shapes/dimensions

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)
```

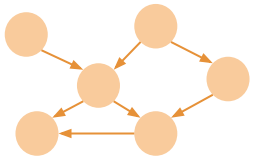
Collect Constraints

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

Solve Constraints

Z3

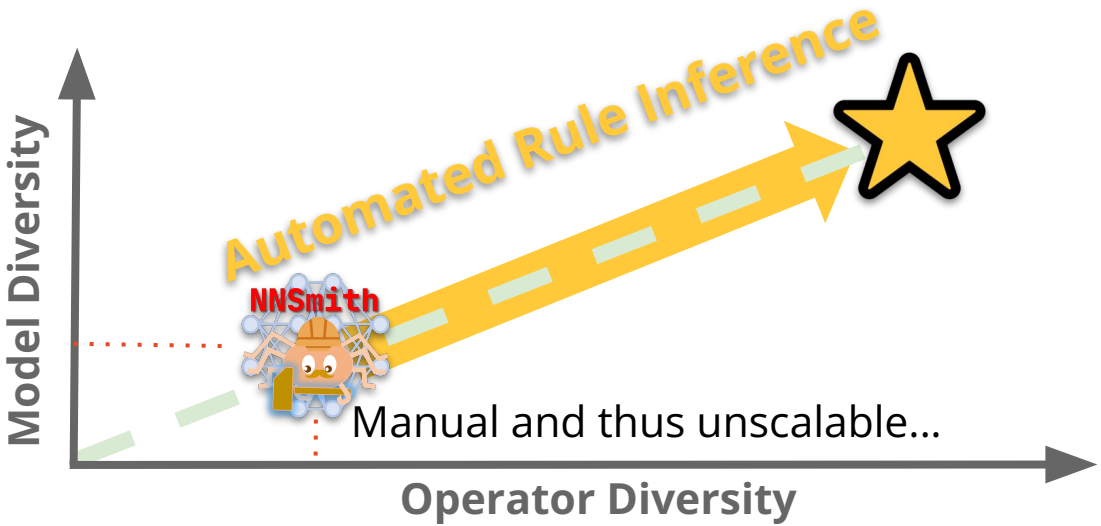
```
{x0=1, x1=8, x2=8, ksize=3,...}
```



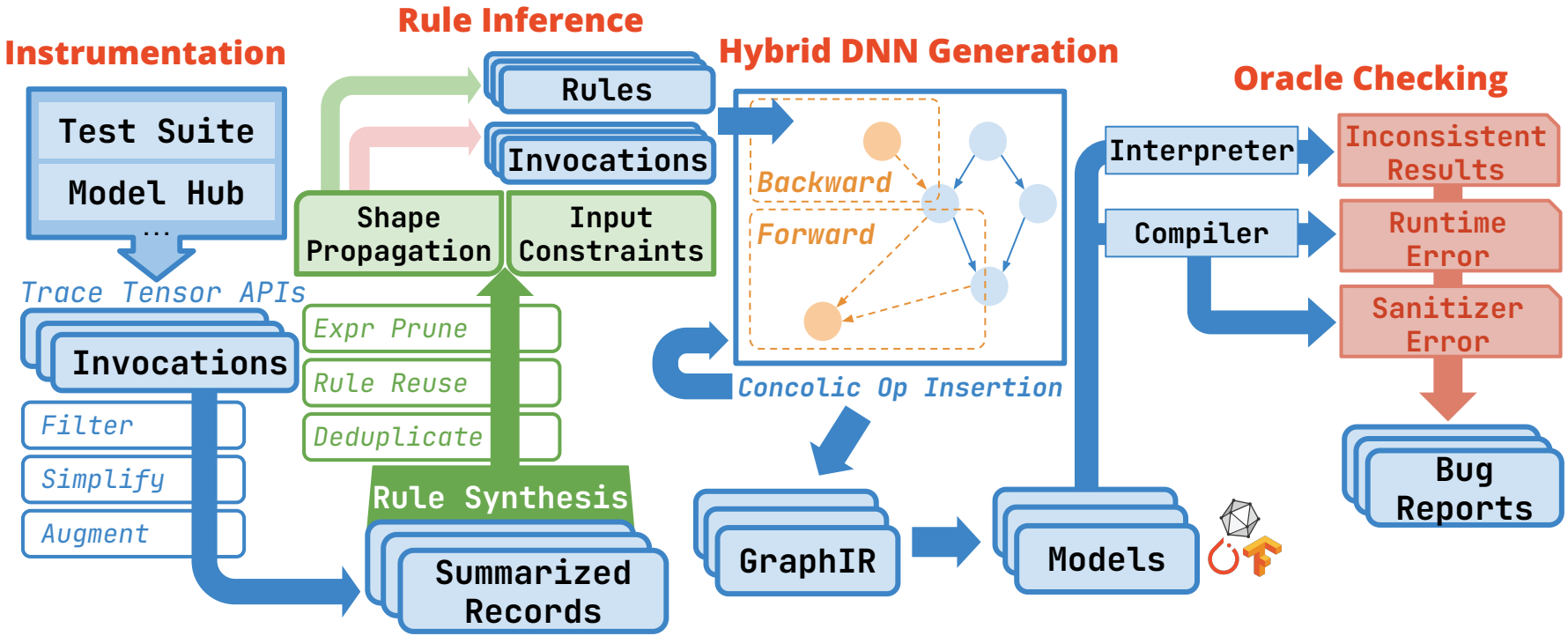
Concrete DNN

Diversifying Valid Models

- **Model diversity** is determined by **operator diversity**
- NNSmith manually supports **~60** operator rules
- Can we *automatically* synthesize operator rules?

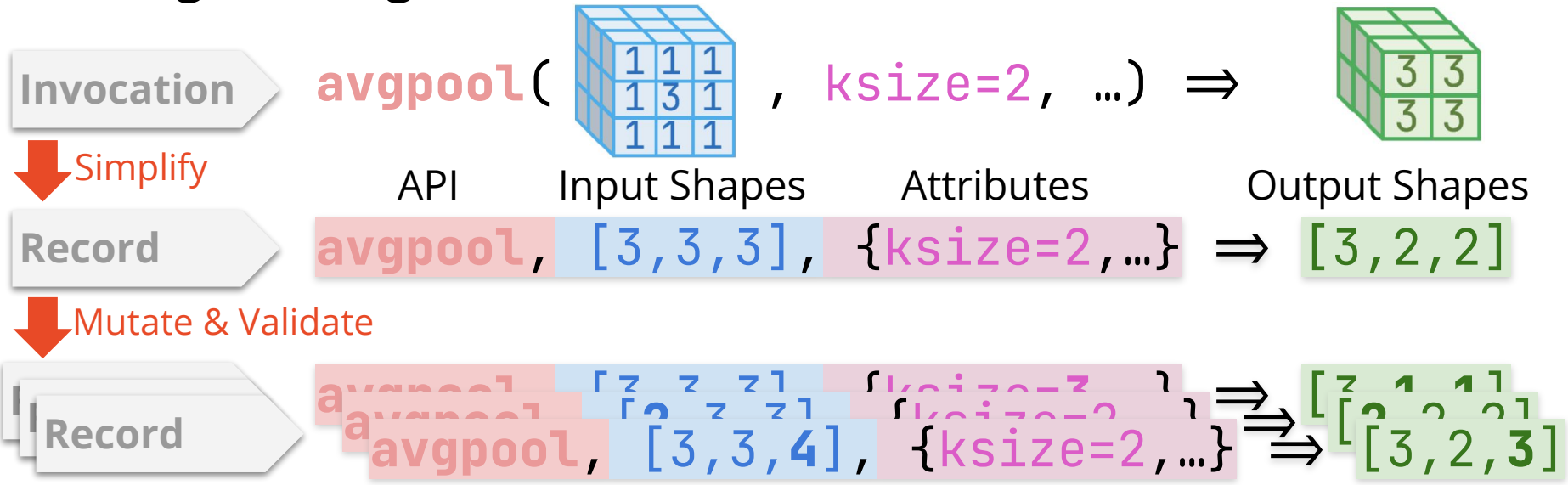


Diversifying Valid Models with NeuRI



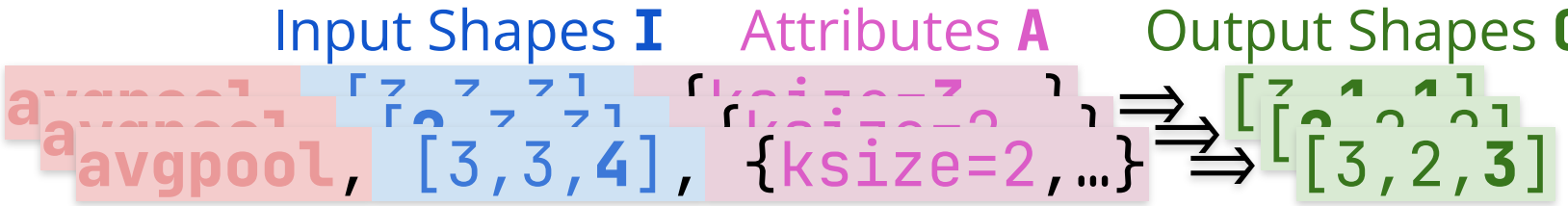
Instrumenting Concrete Operator Invocation

- Instrument operator invocations from regression tests
- Simplify the layout of invocation records
- Augmenting records via mutation



Inferring Operator Rules from Records

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



- **#1 Shape propagation:** `{o = f(A U I); o \in O}`
- **#2 Input constraints:**
  - Equality: `{0 = f(A U I); …}`
  - Inequality: `{0 < f(A U I); …}`
- **Goal:** How to infer `f(A U I)`?

Inductive Rule Inference

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

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

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

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
- Skip **constant sub-term** ⟨op⟩ ⟨literal⟩⟨literal⟩
- **Rarity**: 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

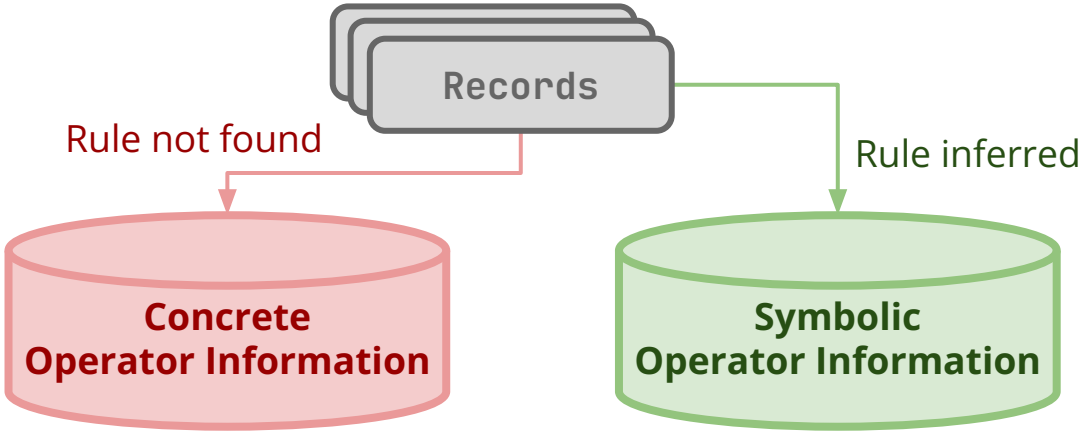
- **Rule reusing**
  - **Insight:** Operator rules can share similar patterns
  - Before rule synthesis, try if the records match any of the inferred rules
- **Post deduplication**
  - Inferred constraints are boilerplate: (i) not readable and (ii) inefficient when used in online solving
  - Example: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

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

`C = C-{c}` *iff* **`conj[C] ⇔ conj[C-{c}]`**

until a fixed point

Concolic Model Generation



NNSmith: Symbolic Graph Generation

- Sketch a whole graph of **symbolic** operators
- Collect and solve constraints over the whole graph
- Cannot make use of **concrete** operator information

Concolic Model Generation (Cont'd)

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

- Constructing a graph <= Inserting an operator
- Inserting a **symbolic** operator
  - Solve the constraints immediately after insertion and materialize it
- Inserting a **concrete** operator
  - Find operator producers/consumers with exact shape match

Evaluation Setup

Systems under Test

- Torch Inductor
- Torch JIT

- 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
- 🔥 **8** bugs are marked as PyTorch high priority
- 🔥 **1** security vulnerability (unpublished yet)

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)

Result Highlights

- **24%** / **15%** coverage improvement over NNSmith
- **95%** / **99%** generated (5-node) models are valid
- **126ms / 70ms** on avg. to generate and run a model
- **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 insightful 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
- Finding **100** bugs including high-priority & security ones!
- Everything open-sourced!

Paper

Artifact

Bug reports

Discussion: CEGIS v.s. NeuRI?

- CEGIS:
  a. **E <- Counter examples**
  b. Rule <- Inductive synthesis over E
  c. **Verify Rule**; if fail E += {counter example} and go to a.
- NeuRI
  a. **E <- Passing/counter examples ahead of time (via instrumentation & mutation)**
  b. Rule <- Inductive synthesis over E
  c. … **verifier not available** for Operator Rules… so we are done here